TABLE DES MATIÈRES

AVANT-PROPOS

Qu'on le veuille ou non, la carte à puce n'est décidément plus la « chasse gardée » d'un club très fermé de professionnels !

N'importe quel PC peut désormais être équipé d'un lecteur de cartes à puce aussi facilement que d'une souris, et pour à peine plus cher, tandis que se démocratisent des lecteurs de poche faisant entrevoir le contenu des cartes les plus populaires.

Des outils de développement sur PC, puissants et gratuits, viennent en même temps faciliter la création de logiciels capables de lire, voire écrire, dans la plupart des cartes en circulation.

Certains donnent d'ailleurs à tout un chacun les moyens de produire ses propres cartes, parfois très spéciales !

Il n'en faut pas davantage pour explorer la « face cachée » des différentes applications « carte » que nous utilisons dans la vie courante, et leur découvrir peut-être des possibilités insoupçonnées…

Bien mieux, l'utilisateur final de cartes à puce dispose ainsi du nécessaire pour évaluer lui-même, en toute objectivité, le degré de confiance qu'il peut raisonnablement leur accorder ; et après tant d'années de « culte du secret », sinon de désinformation, c'est tout simplement révolutionnaire !



CHAPITRE PAGE

1 LES LECTEURS DE CARTES À PUCE

Qu'il soit connecté à un micro-ordinateur, intégré à un appareil quelconque, ou complètement autonome, le « lecteur » est le point de passage obligé pour lire ou écrire dans n'importe quelle carte à puce.

Bien plus qu'un simple connecteur, le lecteur de cartes à puce est un « terminal » intelligent, qui prend plus ou moins largement en charge les protocoles de communication de « bas niveau » avec les cartes.

1.1 CONSTRUIRE OU ACHETER ?

Il n'y a encore que bien peu d'années, la question ne se posait même pas, du moins pour le « simple particulier » : on ne trouvait tout bonnement pas de lecteurs de cartes à puce dans le commerce de détail !

Des dizaines de milliers de passionnés ont donc emboîté le pas à l'auteur de ces lignes qui, dans son ouvrage *Cartes à puce – Initiation et applications* a publié, quitte à faire grincer quelques dents, les premiers plans de lecteurs à construire soi-même.

Outre un prix de revient négligeable, cette approche présentait, et présente toujours, un certain nombre d'avantages.

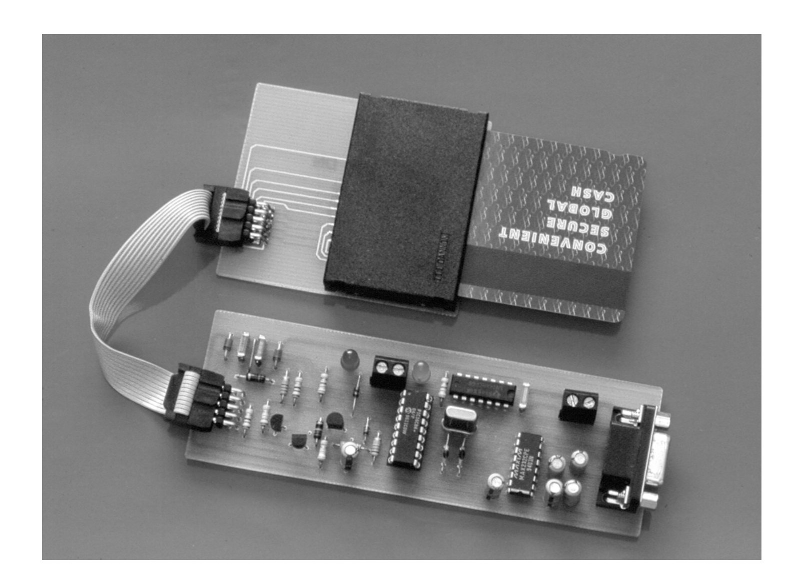
Le principal est sans doute que l'on opère au plus bas niveau possible en matière de protocoles de communication, puisque l'on gère les signaux électriques aux bornes mêmes du connecteur de carte.

On peut ainsi observer tout ce qui s'y passe, y compris un certain nombre de singularités de telle ou telle carte, mais aussi faire varier librement toutes sortes de paramètres au-delà de ce qui est prescrit par les spécifications ; cela pour la bonne et simple raison que le logiciel pilotant le lecteur est entièrement personnalisable, et en aucune façon « à prendre ou à laisser ».

Il doit donc être parfaitement clair que les lecteurs « maison » ont, et auront toujours, leur place dans l'arsenal d'outils d'un bon explorateur de cartes à puce, ou lorsqu'il s'agira de faire lire des cartes à puce par un montage autonome.

Dans cet ouvrage, nous allons pourtant défricher une piste diamétralement opposée : celle qui consiste à mettre en œuvre des lecteurs de cartes à puce du commerce, intégrés d'origine aux PC ou adaptés après coup.

Les avantages de ce choix sont entièrement différents et ouvrent la voie à des aventures d'une tout autre dimension.





Deux lecteurs « maison » de conception très différente.

Un lecteur de cartes à puce du commerce, associé à son « driver » logiciel, se charge en effet de toutes les « basses besognes », et en particulier du strict respect des protocoles de communication normalisés.

C'est fort appréciable, dans la mesure où ceux-ci ont évolué au fil des années, évidemment dans le sens d'une plus grande complexité, rendant à la fois plus délicat et plus fastidieux le développement d'applications ambitieuses sur des lecteurs « maison ».

Le lecteur « tout fait » permettra d'accéder, en toute sécurité, à des cartes très diverses, de façon largement « transparente », depuis un langage de programmation simplement doté de quelques instructions spécialisées, relativement simples à mettre en œuvre.

Bref, en affranchissant le développeur de toute la gestion (fût-elle passionnante !) des protocoles de bas niveau, il permet à celui-ci de concentrer tous ses efforts sur la partie « applicative » de son projet, et donc d'aller beaucoup plus loin, infiniment plus vite, sans se disperser.

Ajoutons enfin que nos fidèles ports série et parallèle pourraient bien disparaître un jour des PC, tandis que certains systèmes d'exploitation s'ingénient déjà à décourager leur manipulation directe…

1.2 PC/SC : VERS DES LECTEURS « GÉNÉRIQUES »

Aussi aberrant que cela puisse paraître, puisque la carte à puce est une invention française, on aura beaucoup « traîné les pieds » en France avant de commercialiser des lecteurs de cartes à puce auprès du grand public (leur prohibition aurait même été à un moment envisagée).

C'est donc vers l'étranger (Allemagne, Grande-Bretagne, Asie) qu'il a fallu se tourner pour dénicher les lecteurs, à la fois performants et peu coûteux, qui sont maintenant disponibles au détail.

Bien que d'excellents lecteurs soient depuis longtemps produits en France (et d'ailleurs largement vendus à l'exportation !), il semble donc évident que l'énorme marché de l'équipement des PC grand public profitera surtout à des fabricants asiatiques. Comprenne qui pourra !

Quoi qu'il en soit, il faut aujourd'hui considérer le lecteur de cartes à puce pour PC comme un produit « générique » : peu importent la marque et le modèle, pourvu qu'il soit certifié « PC/SC ».

En clair, tout logiciel développé dans le respect de la spécification PC/SC pourra (en principe !) fonctionner sur n'importe quel lecteur PC/SC convenablement installé, avec les bons drivers.

Bien entendu, l'auteur de ce livre adhérant avec enthousiasme à cette démarche, la plupart de ses logiciels sont désormais compatibles PC/SC.

À ce point de notre exposé, il faut bien comprendre que, fruit d'une initiative rassemblant les grands noms de la carte à puce et

de l'informatique autour de Microsoft, la compatibilité PC/SC ne prend vraiment tout son sens qu'en environnement Windows 32 bits.

Même si cette spécification est théoriquement ouverte à d'autres systèmes d'exploitation (Mac, Linux, etc.), PC/SC est en effet surtout l'arme secrète de Windows pour s'approprier la carte à puce et ses lecteurs.

Certaines versions de Windows supportent ainsi de façon « native » les fonctionnalités « carte à puce », tandis que la mise à niveau de Windows 95, 98, ou Millenium nécessite simplement l'installation des « Microsoft Smart Card Base Components », disponibles gratuitement ou même présents (dossier SCARD) sur le cédérom d'installation du système.

La **figure 1.1** montre qu'il s'agit simplement d'une « couche » logicielle supplémentaire, venant s'intercaler entre Windows 32 bits et les « drivers » spécifiques d'un ou plusieurs lecteurs réputés compatibles PC/SC.

La spécification PC/SC prévoit en effet un mécanisme censé permettre la coexistence paisible de plusieurs lecteurs et de plusieurs applications faisant appel, ensemble ou séparément, à une ou plusieurs cartes à puce (asynchrones, I2C, ou à protocole 2/3 fils).

Figure 1.1.
Le principe de la compatibilité PC/SC.

En pratique, l'expérience montre que dans l'état actuel des choses, il est prudent d'éviter d'associer des lecteurs PC/SC de différentes marques, car la bonne entente entre drivers concurrents est encore loin d'être générale !

Une liste à jour des lecteurs certifiés PC/SC est disponible sur le site http://www.pcscworkgroup.com.

On y trouvera des modèles populaires de chez Gemplus, aux côtés de produits pas toujours très connus en France.

Deux exemples assez représentatifs sont l'ACR 30 de chez ACS (disponible au détail auprès de Selectronic) et l'ACR 20 « CyberMouse » (http://www.hitechtools.com).



« CyberMouse » est un excellent lecteur PC/SC !

À signaler également le ChipDrive Micro de chez Towitoko (http://www.towitoko.de ou http://www.scmmicro.com), disponible auprès de Conrad Elec-tronique ; cela pour les lecteurs se branchant sur un port série, car il existe aussi des lecteurs PC/SC pour port USB, port clavier (dit PS2), ou port PCMCIA.

Moyennant l'installation (réussie !) des bons drivers, une application développée dans l'esprit PC/SC doit fonctionner de façon « transparente » sur tout lecteur PC/SC, quel que soit le port matériel qu'il utilise.

Notre ouvrage *Téléphones portables et PC*, paru dans cette même collection, en fournit la parfaite illustration : à partir de sa seconde édition (août 2002), son cédérom contient toute une collection d'utilitaires PC/SC pour cartes SIM !

Mais la plupart des lecteurs PC/SC disposent également d'un mode de fonctionnement « propriétaire », plus ou moins sévèrement incompatible avec le mode PC/SC.

Il peut servir (et nous ne nous en priverons pas) à gérer des cartes ou des fonctionnalités non supportées par la spécification PC/SC, ou bien, et c'est nettement moins glorieux, à empêcher l'utilisation, avec un lecteur concurrent, d'un logiciel servant de « produit d'appel ».

D'une façon générale, si le driver PC/SC d'un lecteur est installé, il s'activera spontanément si le lecteur est déjà connecté au PC lors de sa mise en route (comportement « Plug & Play »). Cela contrariera alors bien souvent l'exécution des logiciels compatibles avec le seul mode « propriétaire ».

À l'inverse, si le lecteur n'est connecté qu'après le démarrage réussi de Windows (par exemple avec un commutateur RS232), alors il fonctionnera très probablement dans son mode « propriétaire », et ne sera pas reconnu par les applications PC/SC.

Cela dit, il est fréquemment possible de faire fonctionner simultanément un lecteur PC/SC sur un port donné, et un lecteur « propriétaire » sur un autre, ou encore deux lecteurs PC/SC reliés à des ports différents.

Dans ce dernier cas, le logiciel peut proposer de choisir entre les lecteurs disponibles, ou bien sélectionner d'autorité un lecteur « par défaut » : une inépuisable source de surprises…

1.3 DES LECTEURS DE POCHE

Si les lecteurs de cartes à puce sont en passe de devenir des périphériques de PC aussi courants que les souris, on commence aussi à en trouver dans les poches de monsieur Tout-le-monde !

À côté des lecteurs dédiés à tel ou tel porte-monnaie électronique ou à diverses cartes privatives, les modèles « génériques » compatibles avec certains outils de développement ont de quoi séduire le développeur imaginatif.

Comme bien des idées géniales, c'est en France qu'a germé celle d'un lecteur de cartes à puce autonome, et au format de poche.

Développé par le SEPT (le regretté département R&D commun de La Poste et de France Télécom), le prototype de la « Cartulette » avait déjà fait sensation au salon CARTES 91.

Permettant, dès cette époque, de consulter l'historique des transactions d'une carte bancaire, ou de lire le solde d'unités d'une

télécarte, cette maquette avait pour objectif plus lointain de supporter le « PME », autrement dit le Porte-monnaie électronique, dont La Poste nourrissait l'ambition d'être un acteur majeur.

Quelques années plus tard, une version à coupleur acoustique incorporé était imaginée : l'ancêtre, en somme, de bien des produits d'authentification par téléphone d'aujourd'hui !

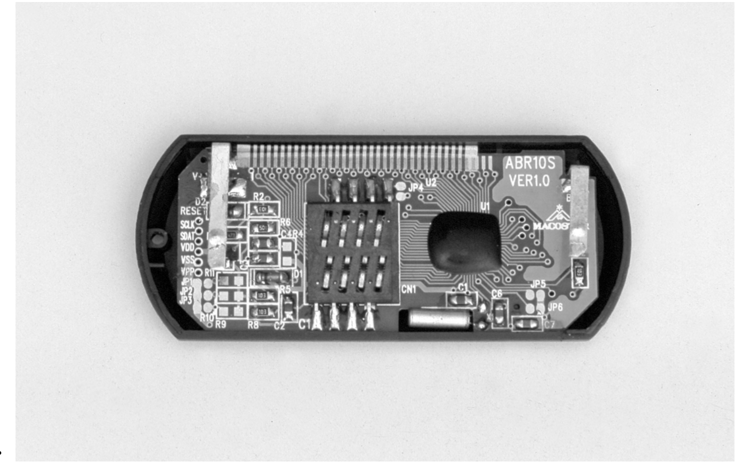
Maintenant, le lecteur de cartes à puce de poche se présente sous la forme ultra-miniaturisée d'un porte-clefs, ou bien d'un étui dans lequel la carte peut rester rangée.

De par ses dimensions plus généreuses, cette seconde présentation permet de mettre en œuvre des fonctions de type « calculette », au moyen d'un clavier et d'un afficheur à plusieurs lignes.

La fabrication de masse est devenue très peu coûteuse grâce, notamment, à la technologie *chip on board*. Elle est, bien entendu, réalisée presque uniquement en Asie, soit selon des plans conçus en Europe, soit à partir de développements locaux.

C'est ainsi que le fabricant ACS (*Advanced Card Systems*) conçoit et produit à Hong Kong ses lecteurs de poche « ABR », offrant de larges possibilités de personnalisation en fonction des besoins de chaque application.

Bien que basé sur cette même plate-forme matérielle, le « Pocket Reader » de certains kits de développement bouscule complètement cette approche, puisqu'il se veut résolument « générique ». En clair, ce n'est pas le lecteur qui s'adapte (industriellement) à telle ou telle carte, mais la carte qui s'adapte (par simple programmation) à ce lecteur « à prendre ou à laisser ».



Vue interne d'un lecteur « porte-clefs » ABR-1.

Naturellement, nous n'allons pas nous priver d'exploiter (au chapitre 6), cette opportunité dont on ne mesure pas forcément très bien la portée au premier abord…

Il est important de remarquer que la production de masse, et donc à bas prix, de lecteurs de poche n'a véritablement pris son essor que grâce à l'apparition de microcontrôleurs à interface « carte à puce » incorporée.

Même si des concurrents arrivent maintenant sur le marché, c'est à STMicroelectronics (grand spécialiste des composants encartables) que l'on doit l'un des premiers « monochips » pour lecteurs de cartes à puce : le ST 72411 R.

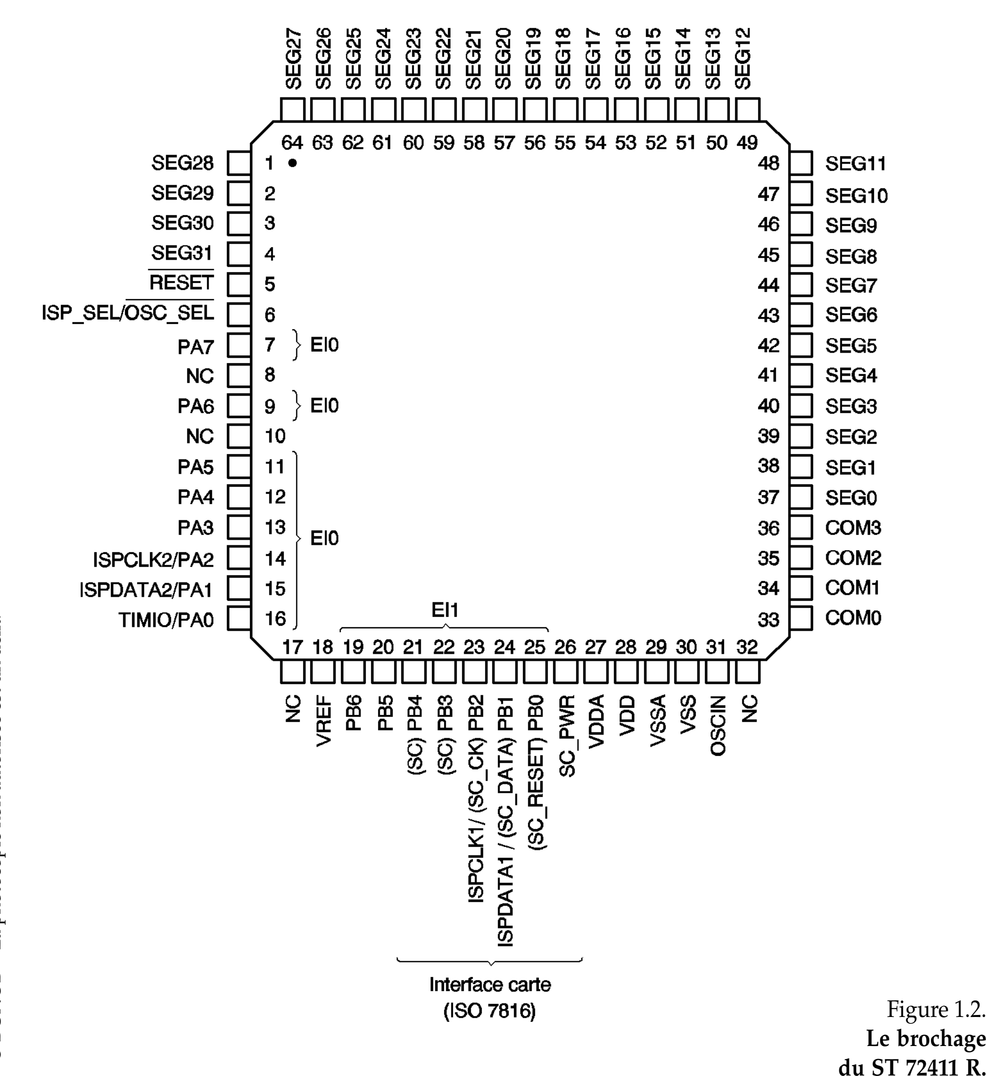


Figure 1.2.
Le brochage du ST 72411 R.

D'après son brochage, reproduit à la **figure 1.2**, on voit du premier coup que ce membre de la famille ST7 dispose tout à la fois d'une interface directe pour afficheur LCD (4 lignes de 32 caractères alphanumériques), de six lignes d'interface ISO 7816 pouvant rejoindre directement un connecteur de cartes à puce (asynchrones et/ou synchrones), et de ports d'entrée-sortie capables de gérer, par exemple, un clavier organisé en matrice.

Différents modes « basse consommation » et des possibilités de réduction de la fréquence d'horloge « carte » lui permettent de se contenter de deux piles « bouton » au lithium, tout en supportant aussi bien les cartes à puce 5 V (classe A) que 3 V (classe B).

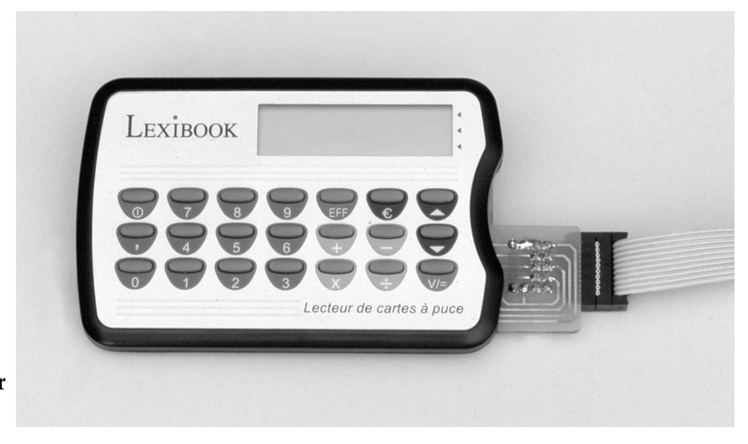
Avec 4 Ko de ROM (ou de Flash !) et 256 octets de RAM, ce genre de processeur, compatible avec les puissants outils de développement de la famille ST7, se prête à la réalisation de lecteurs de poche aux fonctionnalités assez ambitieuses.

C'est ainsi que sont apparus, courant 2002, les derniers modèles de Xiring (commercialisés aussi sous la marque Lexibook), les XL 2500.

Par rapport aux très populaires XL 2000, déjà capables de lire la carte bancaire, la carte Vitale, et les télécartes de 1re et 2e générations, des fonctions de lecture du porte-monnaie électronique « Monéo » ont été ajoutées.

De quoi commencer à mettre le système à l'épreuve (voir chapitre 6), avant même que les cartes ne nous soient proposées (ou imposées…) par les banques !

Mais en attendant ce très médiatique « porte-monnaie électronique », les premiers utilisateurs de lecteurs de poche ont été… les collectionneurs de télécartes !



Un lecteur de poche XL 2500 prêt à révéler ses « petits secrets ».

Pendant qu'un peu partout dans le monde, les établissements financiers distribuaient des lecteurs de PME Proton, Visacash, GeldKarte, Mondex, etc., des négociants en cartes de collection lançaient, en France, l'idée de ces porte-clefs « testeurs » de télécartes, qui ont véritablement « fait un malheur » dans les milieux concernés.

En pratique, des applications aussi fondamentalement différentes peuvent mettre à contribution, si elle est bien conçue, une seule et même plate-forme matérielle.

Il suffit, en effet, de développer le logiciel approprié, puis de le charger dans la mémoire Flash du lecteur, quitte à passer à une version ROM si les volumes devaient le justifier.

1.4 DES TERMINAUX AUTONOMES

À mi-chemin entre les lecteurs de poche et les applications « tournant » sur un PC, il y a place pour toute une variété d'appareils dotés de fonctions « carte à puce » plus ou moins élaborées.

À cent lieues des incontournables terminaux de paiement, les « copieurs de cartes SIM » sont un bon exemple de machines relativement simples, dotées pourtant de deux lecteurs de cartes à puce.

En effet, face au problème que pose le transfert des données personnelles (annuaires, messages, etc.) lors du renouvellement de la carte SIM d'un téléphone portable (ou en cas de changement d'opérateur…), il a bien fallu imaginer de petits appareils capables d'automatiser l'opération.

Alors même que les opérateurs voient toujours d'un très mauvais œil que leurs clients « bricolent » leurs cartes SIM, certains fabricants semblent compter sur eux pour leur offrir un copieur en même temps que leur nouvelle carte.

D'autres, comme ACS, visent plus sagement le marché des points de vente, souvent dépourvus de moyens simples et rapides pour transférer le contenu d'une carte SIM vers une autre, en toute confidentialité ou même en « libre-service ».

À vrai dire, il semblerait que les lecteurs de nos livres soient souvent mieux équipés que les boutiques des opérateurs, d'autant qu'ils travaillent volontiers à l'aide d'un lecteur connecté à un PC.

L'« ACR SIM copier » bénéficie de toute l'expérience acquise par ACS avec son logiciel « SIMmate 2000 » (voir notre ouvrage *Téléphones portables et PC*), dont il intègre en quelque sorte une version fortement allégée, extrêmement simple à manier.



Le copieur de cartes SIM d'ACS.

En fait, ce sont peut-être les revendeurs indépendants des opérateurs qui apprécient le plus ce genre d'innovation. Ils se trouvent, en effet, au cœur même du phénomène de « churn », autrement dit de nomadisme d'un opérateur vers un autre, qu'ils sont d'ailleurs parfois amenés à encourager.

1.5 UN LECTEUR AUTONOME « MAISON »

En l'absence d'objectifs de miniaturisation et d'industrialisation à bas prix, il est parfaitement possible de s'inspirer du concept de « lecteur de poche » autonome, tout en mettant en œuvre des composants à usage général.

C'est ainsi que le présent projet fait appel à un très classique PIC 16F84, programmé en Basic.

Nous avions déjà amplement vanté, dans notre ouvrage *Basic pour microcontrôleurs et PC*, les mérites de cette approche hautement simplificatrice.

Depuis, des compilateurs encore plus performants ont été mis sur le marché, accompagnés de puissantes bibliothèques de « périphériques virtuels » tels qu'un afficheur LCD.

C'est ainsi que même la version de démonstration, sévèrement limitée en longueur du code source, du compilateur LET Basic (distribué en France par Selectronic) suffira pour doter notre maquette de possibilités déjà significatives.

Bien entendu, chacun pourra ensuite étoffer cet exemple en se servant de la version complète, afin de le doter de telle ou telle fonctionnalité particulière : la voie sera toute tracée !

Attardons-nous donc tout d'abord un peu sur les possibilités de ce compilateur, qui offre même la possibilité rare de produire, entièrement sous environnement Windows, du code pour les très populaires PIC 12C508 et PIC 12C509 (à noyau 12 bits, rappelons-le).

Fonctionnant en tandem avec l'assembleur « officiel » MPASM de Microchip, il permet d'examiner le code assembleur intermédiaire, et par conséquent d'en apprécier objectivement la qualité.

C'est la firme britannique Crownhill Associates, d'ailleurs très active dans le domaine des cartes à puce, qui édite ce compilateur.

Le cédérom de cet ouvrage contient (dans le dossier « TERMIN ») deux versions « Lite » assez représentatives de l'évolution du produit :

- la version 7.00 (setup700.exe) supporte de nombreuses références de PIC, dont les 12C50x (ce dont nous ne nous priverons pas de profiter un peu plus loin !).
  Comme toutes les versions intermédiaires de n'importe quel logiciel, elle souffre toutefois encore de quelques imperfections parfois délicates à contourner ;
- la version 7.10 (setup710.exe), très sensiblement améliorée, ne supporte plus, en revanche, que le PIC 16F84.

Les versions ultérieures pourront être téléchargées sur le site http://www.letbasic.com sous la forme d'un unique fichier nommé setup.exe.

L'exécution de ce fichier entraîne l'installation du logiciel et d'une version récente de MPASM pour Windows, le tout étant accompagné d'un manuel très complet en format PDF.

Celui-ci est également reproduit (dans sa version 7) sur notre cédérom, et ses versions successives seront aussi téléchargeables sur le même site.

On s'apercevra immédiatement que cet outil se démarque clairement du PBasic (Parallax Basic) hérité du Basic Stamp, pour revenir à un dialecte beaucoup plus généraliste de ce langage à tout faire.

L'effort d'adaptation sera ainsi nettement moindre pour les habitués de GWBasic ou de QBasic (voire de ZCBasic, nous y reviendrons !), bien que de multiples instructions nouvelles fassent leur apparition.

Bon nombre d'entre elles concernent les « périphériques virtuels » fournis sous la forme de bibliothèques à inclure et initialiser si et seulement si on en a besoin :

- gestion d'afficheurs LCD ;
- support du bus I2C ;
- support d'une ligne série 9 600 bauds (voire d'une carte à puce asynchrone…) ;
- gestion d'un clavier à 16 touches ;
- conversion analogique-numérique avec le PIC 16C71.

Précisons enfin qu'il est possible de « mélanger » hardiment Basic et assembleur dans le code source, ce qui démultiplie encore les possibilités offertes au programmeur maîtrisant les deux langages.

C'est plus qu'il n'en faut pour développer un lecteur matériellement capable de prendre connaissance du contenu de n'importe quelle carte à puce synchrone, et de l'afficher séquentiellement sur un écran LCD à deux lignes de 16 caractères.

Dans la version logicielle présentée ici (programme I2CPIC.BAS), il est prévu de se limiter aux cartes à puce I2C de 2 kbits (soit 256 octets) telles que la D2000 Philips (disponible au détail chez Selectronic) ; cela afin de profiter d'un second périphérique virtuel, capable de gérer de façon transparente l'accès à une mémoire I2C externe, par ses lignes SCL et SDA (horloge et données).

Le schéma de la **figure 1.3** montre toutefois que le PIC 16F84 (ou 16C84) communique avec les autres contacts du connecteur de carte, d'où la possibilité de reprogrammer le montage pour lire à peu près n'importe quelle sorte de carte synchrone.

Nous verrons, au chapitre 3, comment prendre connaissance du protocole de communication de ce genre de carte, même inconnue, en vue d'en programmer les principales « micro-instructions » sous la forme de quelques lignes de Basic ; cela au prix de l'écriture d'un logiciel forcément plus long, et exigeant donc le recours à la version payante du compilateur.

Pour l'heure, le code source que voici affiche le résultat de la lecture sous la forme de seize groupes de seize octets, convertis chacun en un caractère déduit du code ASCII correspondant.

L'expérience montre, en effet, que bien des cartes I2C contiennent en réalité des caractères ASCII : cartes-jeu longtemps distribuées par Gemplus au salon CARTES, « Password Card » (de Towitoko) utilisée pour mémoriser les mots de passe d'un utilisateur de PC, etc.

Chaque bloc de caractères est affiché pendant une seconde et demie (six fois le maximum de 255 ms autorisé par le compila-

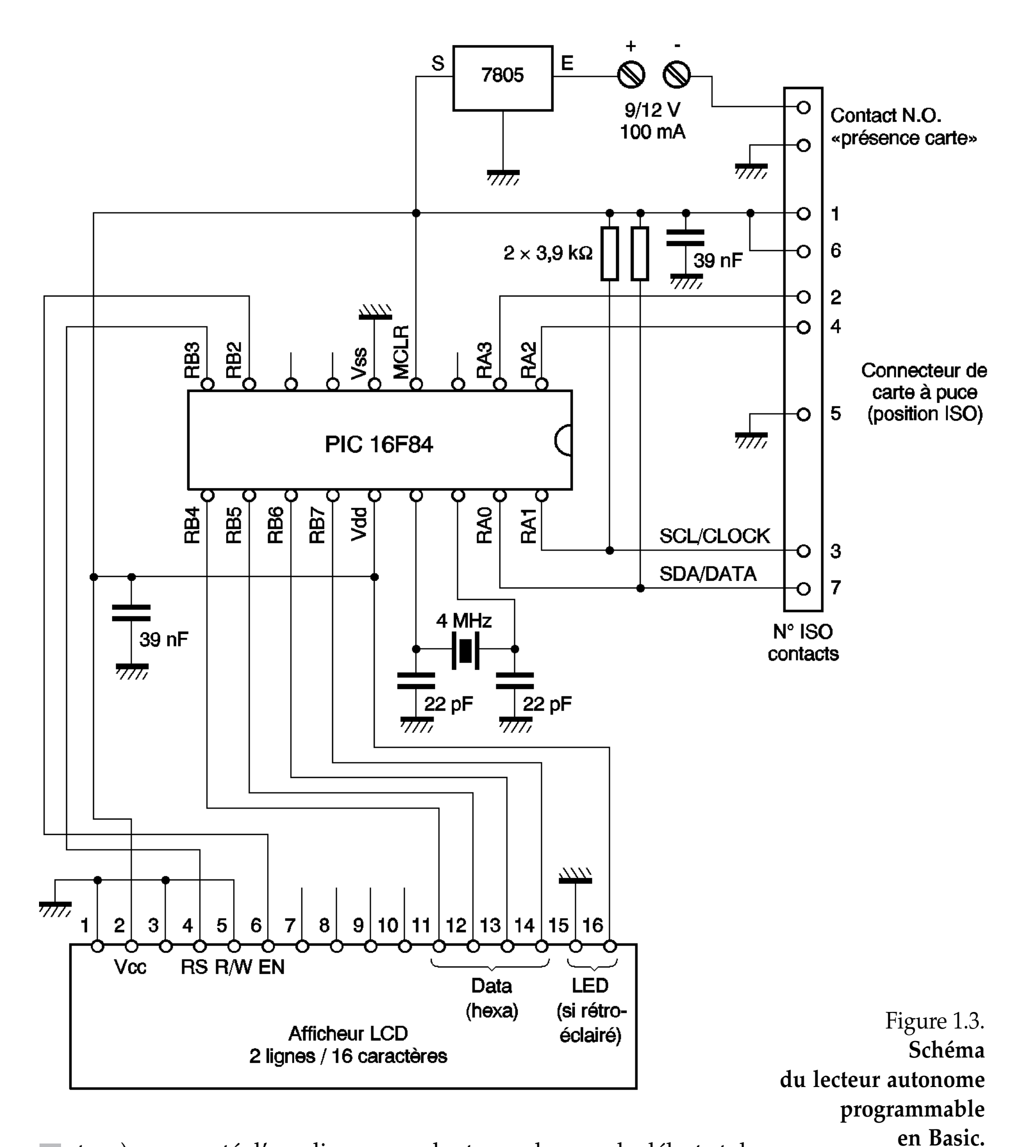


Figure 1.3.
Schéma du lecteur autonome programmable en Basic.

teur), surmonté d'une ligne rappelant ses adresses de début et de fin en hexadécimal (ce qui fournit l'occasion d'expérimenter avec plusieurs formes possibles de l'instruction PRINT).

Le cas échéant, il serait d'ailleurs extrêmement facile de modifier le programme pour que l'affichage se fasse en décimal ou en hexa.

```
DEVICE 16F84
INCLUDE LCD
INCLUDE I2CBUS
DIM A, B, C, F, G, H
INIT LCD,PortB
```

```
INIT I2CBUS
for g=0 to 15
CLS : H = 16*G : C = H+15
PRINT $H," .......... ",$C : CURSOR 1,2
FOR F=0 to 15
B=H+F : A=MEMREAD(B) : PRINT #A
NEXT F
GOSUB tempo : GOSUB tempo : GOSUB tempo
NEXT G
CLS : PRINT "Lecture terminee"
CURSOR 1,2
PRINT "Retirer la carte"
STOP
tempo: delayms(255) : delayms(255) : return
END
```

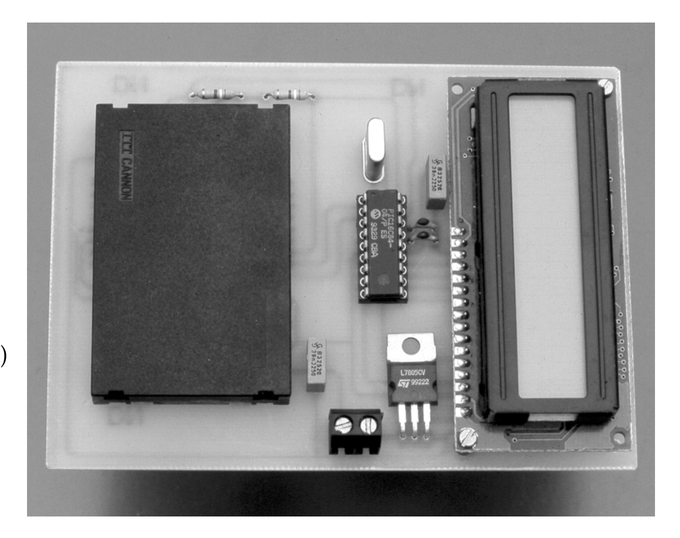
Toute la complexité du montage étant prise en charge par sa partie logicielle, la réalisation pratique se limite à l'interconnexion de fort peu de composants, sur un circuit imprimé gravé d'après la **figure 1.4**.

Le plan de câblage de la **figure 1.5** est prévu pour un afficheur rétro-éclairé, ce qui suppose l'emploi d'un régulateur 7805 pour subvenir à sa consommation non négligeable (au moins 100 mA sous 5 V).

En présence d'un modèle uniquement réflexif, un 78L05 suffirait, tandis que l'afficheur ne serait muni que de 14 connexions au lieu de 16.

On notera que celui-ci fonctionne en mode « hexa », c'est-à-dire sur des blocs de données de seulement quatre bits. Cela minimise

Liste des composants
PIC 16F84 à programmer (I2CPIC.HEX)
Régulateur 7805
Quartz 4 MHz (environ)
Afficheur LCD (2 × 16 caractères)
2 × 3,9 kΩ
2 × 39 nF
2 × 22 pF
Connecteur de carte à puce (CCM01 2NO-3 ITT-CANNON)
Bornier 2 circuits 5,08 mm



Le lecteur autonome de cartes I2C.

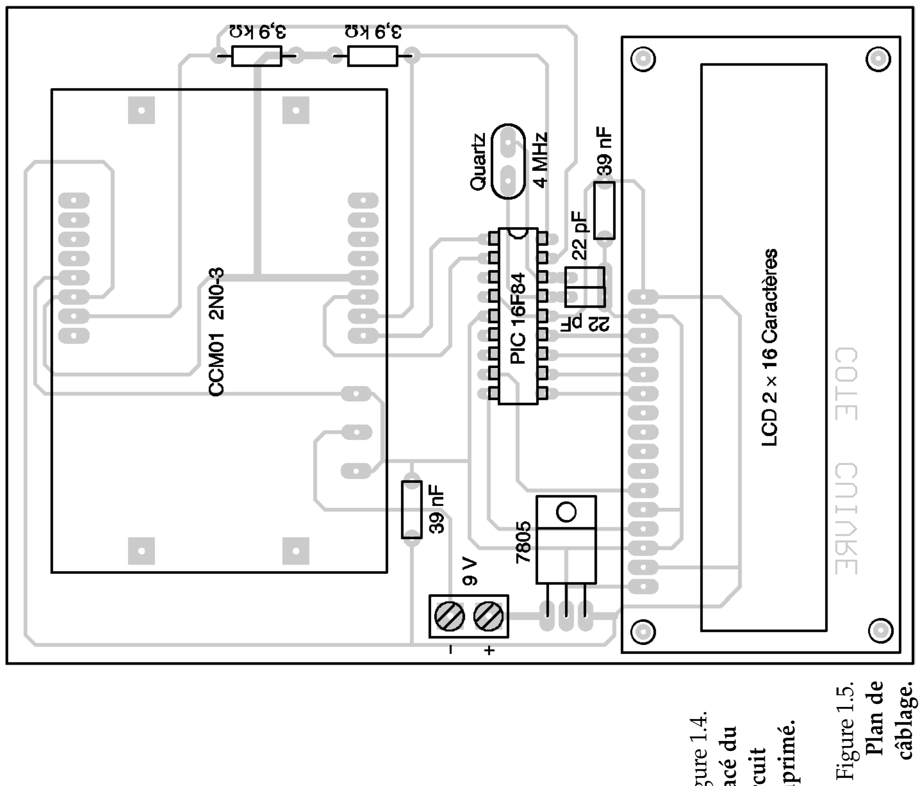


Figure 1.5. **Plan de câblage.**

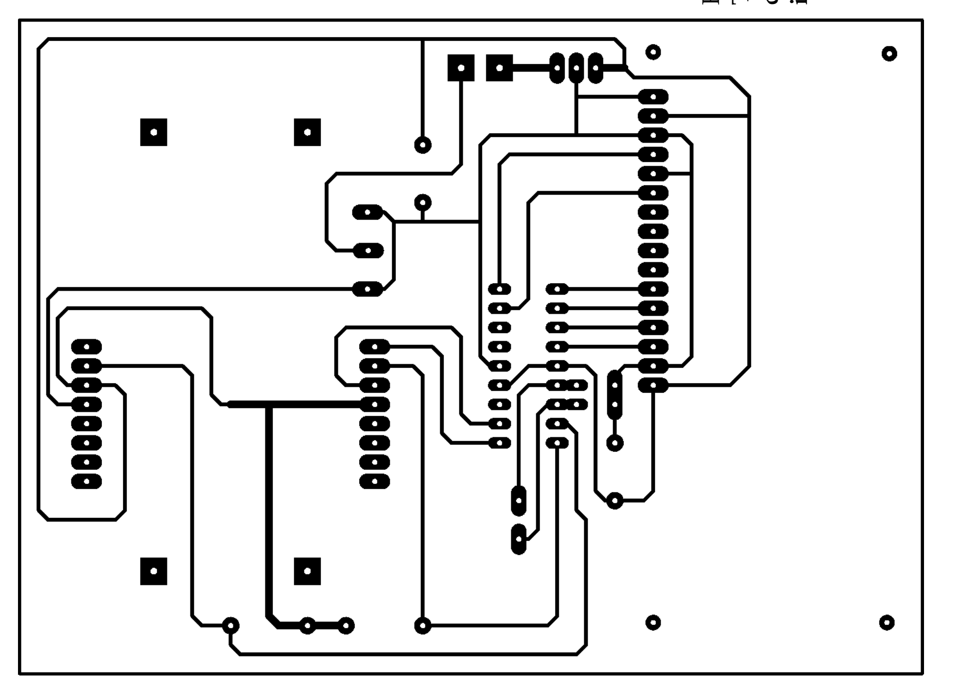
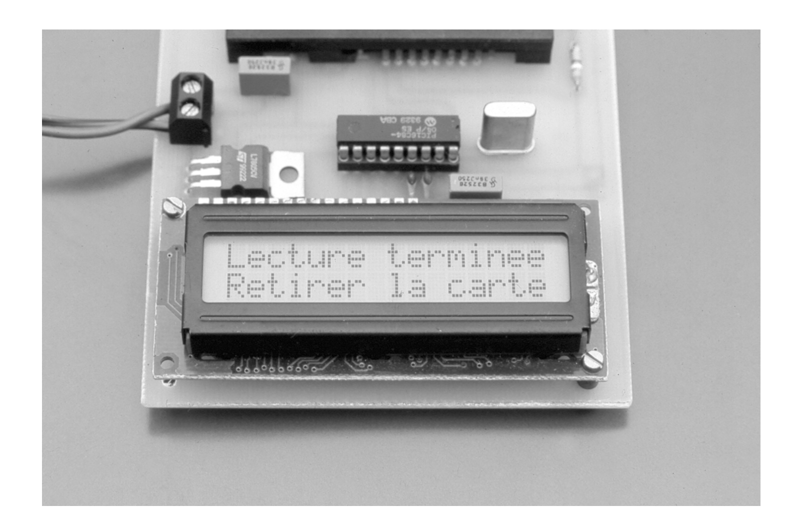


Figure 1.4. **Tracé du circuit imprimé.**





Le lecteur autonome de cartes I2C (suite).

le nombre de broches du PIC utilisées pour la communication, ménageant ainsi la possibilité d'éventuelles extensions du montage à partir des lignes d'entrée-sortie restant disponibles.

C'est là toute la différence avec un microcontrôleur spécialisé, qui utilise couramment jusqu'à 36 lignes pour piloter un afficheur sans « intelligence » incorporée, et donc très largement moins coûteux en production de série.

Le PIC sera programmé au moyen du fichier I2CPIC.HEX produit par le compilateur, et dont notre cédérom contient une version « enrichie » des positions des fusibles de configuration (oscillateur en mode XT, WDT OFF, PWRT OFF, et CP OFF).

On remarquera au passage que l'opération ne consomme que 391 mots de mémoire (soit un gros tiers des 1 024 mots disponibles), dont l'essentiel correspond d'ailleurs aux bibliothèques de gestion de l'afficheur et du bus I2C.

Le PIC sera monté de préférence par l'intermédiaire d'un support à contacts « tulipe », et il ne restera plus qu'à alimenter la maquette sous une tension continue de 9 ou 12 V.

L'alimentation se faisant à travers le contact « présence carte » du connecteur de carte à puce, rien ne doit se passer tant qu'une carte n'est pas introduite.

À ce moment seulement, l'afficheur doit s'éclairer et la lecture peut commencer.

Après les seize étapes de l'affichage (adresses 00h à F0h), un message invitant à retirer la carte s'affichera jusqu'à son extraction effective.

Bien entendu, la carte doit contenir des données pour que quelque chose de vraisemblable s'affiche !

En tant que cartes à lecture et écriture libres, les cartes I2C se prêtent idéalement au transport de petits volumes de données en tous genres, et nous verrons au chapitre 3 comment les remplir depuis un PC.

Bien entendu, le compilateur LET Basic peut aussi servir à développer des programmes gérant l'écriture dans ces cartes : c'est quasiment aussi simple que pour la lecture !

1.6 DES SERRURES À CARTES À PUCE

L'idée d'utiliser des télécartes usagées comme clefs électroniques n'est pas neuve, mais peut se décliner de multiples façons.

Dans notre ouvrage *Basic pour microcontrôleurs et PC*, nous montrions ainsi comment un même développement en PBasic pouvait indifféremment être mis en œuvre sur un Basic Stamp ou sur un PIC 16C84.

Il nous paraît intéressant de montrer ici comment programmer exactement les mêmes fonctions à l'aide du compilateur LET Basic.

Ceux de nos lecteurs qui ne posséderaient pas l'ouvrage en question pourront se référer à la **figure 1.6** pour suivre notre propos.

Le code source ci-après (SMARTLET.BAS), compilé en SMARTLET.HEX, sera chargé une bonne fois pour toutes dans un PIC 16F84, aux fusibles de configuration correctement positionnés, que l'on montera sur un circuit imprimé gravé selon la **figure 1.7**.

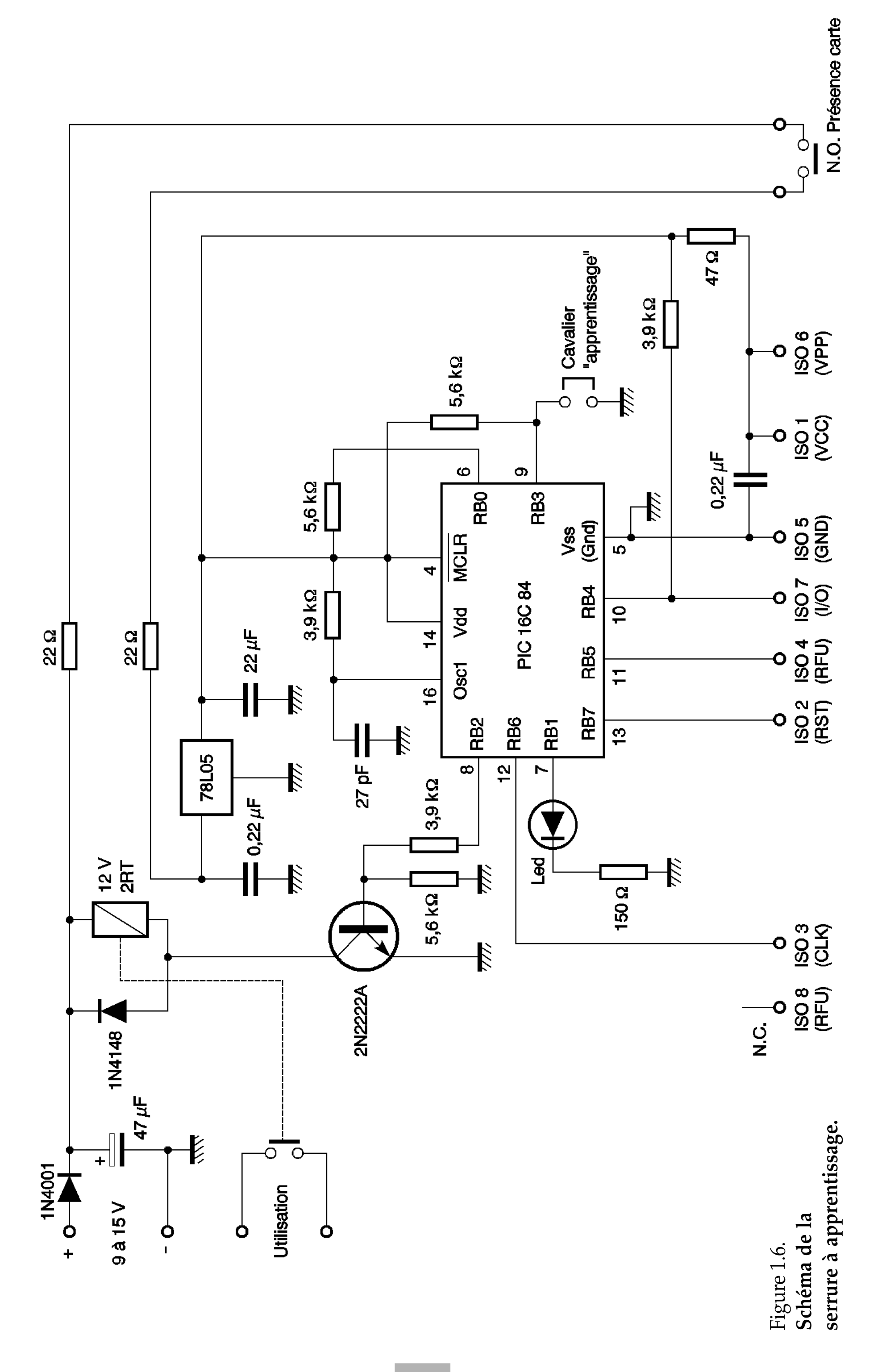


Figure 1.6.
Schéma de la serrure à apprentissage.

```
DEVICE 16C84
INCLUDE EEPROM
INIT EEPROM
DIM B0,B1,B2,B3,B4,S,H
DEFINE PortB=%00011001
OUTB(2) : HIGH B.7 : HIGH B.6 : LOW B.6
LOW B.7 : HIGH B.6 : LOW B.6 : HIGH B.5
H=0 : IF INPORTB & 1 = 0 THEN H=32
IF INPORTB & 8 = 0 then GOTO learn
IF INPORTB & 8 = 8 then GOTO check
learn: FOR B0=0 TO 31
B2=0
FOR B1=0 TO 7
B2=B2*2 : S=INPORTB & 16 : S=S/16 : B2=B2+S : HIGH B.6 : LOW B.6
NEXT B1
STORE B0+H,B2
NEXT B0
OUTB(0) : B4=50 : GOTO Led
check: FOR B0=0 TO 31
B2=0
FOR B1=0 TO 7
B2=B2*2 : S=INPORTB & 16 : S=S/16 : B2=B2+S : HIGH B.6 : LOW B.6
NEXT B1
B3=EEDATA(B0+H) : IF B2 <> B3 THEN GOTO False
NEXT B0
OUTB(6) : STOP
False: OUTB(0) : B4=255 : GOTO Led
Led: FOR B0=0 TO 10
HIGH B.1 : DELAYMS(B4) : LOW B.1 : DELAYMS(B4)
NEXT B0 : END
```

En plus des autres composants, à câbler selon la **figure 1.8**, il faudra ajouter un connecteur pour cartes à puce et un cordon d'interconnexion, le tout réalisé selon les plans fournis au chapitre 2 (Des outils connectiques).

Sans entrer de nouveau dans tous les détails, précisons que cette serrure fonctionne par « apprentissage » :

Le cavalier de commutation étant placé en position « court-circuit », toute télécarte insérée dans le connecteur sera lue, et son contenu mémorisé dans la mémoire non-volatile (EEPROM) du PIC.

Le cavalier étant retiré, toute carte insérée par la suite sera comparée à cette « image » de référence, et le relais ne collera que si la carte originale (ou une carte strictement identique) est reconnue.

Mais nous allons maintenant montrer qu'il est possible de suivre une voie radicalement différente, en utilisant cette fois un PIC 12C508.

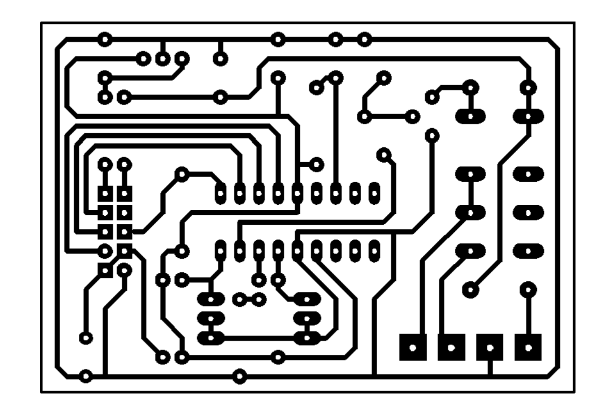


Figure 1.7.
Tracé du circuit imprimé.

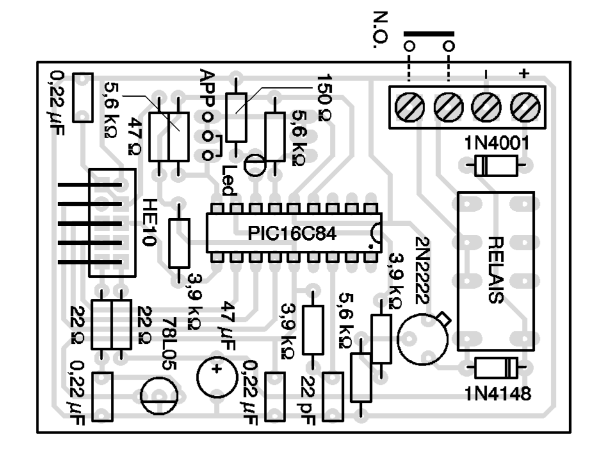
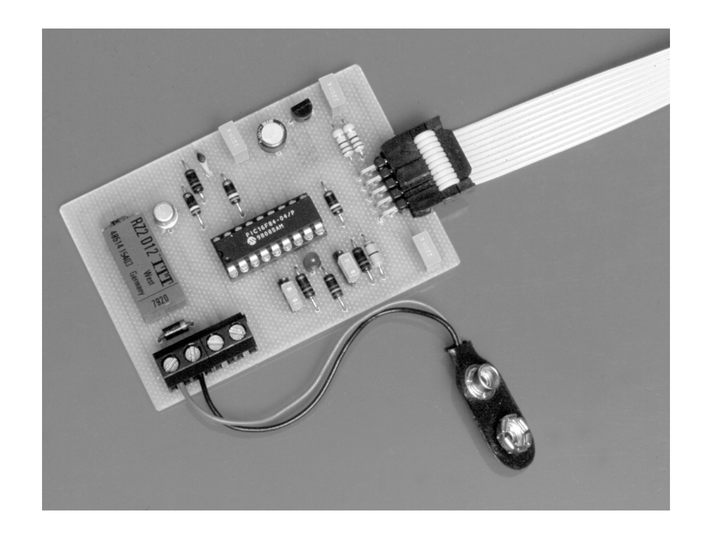


Figure 1.8.
Plan de câblage.

Liste des composants
PIC 16F84 à programmer (SMARTLET.HEX)
1 × 78L05
1 × Led rouge
1 × 2N 2222
2 × 1N 4148
2 × 22 Ω
1 × 47 Ω
1 × 150 Ω
3 × 3,9 kΩ
3 × 5,6 kΩ
1 × 27 pF
1 × 47 µF 16 V radial
2 × 0,22 µF
Relais DIL 12 V 2RT
Bornier 4 circuits 5,08 mm
Barrette sécable à doubles picots carrés coudés
Barrette sécable à picots carrés droits
Cavalier de court-circuit 2,54 mm



Le circuit imprimé principal de la serrure.

Rappelons que ce microcontrôleur, particulièrement économique, ne contient pas d'EEPROM de données, et dispose de beaucoup moins de lignes d'entrée-sortie, ce qui change très sensiblement la donne !

On sait bien que tout l'intérêt d'une serrure à carte à puce réside dans l'infinie variété des codages envisageables à partir de cartes de récupération, éventuellement reprogrammées.

Une simple télécarte usagée, par exemple, renferme un « numéro matricule » unique, facile à lire dans sa mémoire mais très difficile à modifier ou à contrefaire.

Une serrure programmée pour reconnaître cet identifiant ne pourra, par conséquent, être actionnée que par cette seule et unique carte, à moins d'arriver à en fabriquer un « clone » fidèle.

Diverses approches sont envisageables pour que plusieurs « clefs » puissent ouvrir la même « porte » :

- programmer la serrure avec une liste limitative d'identifiants devant être indifféremment reconnus ;
- utiliser des cartes contenant toutes le même identifiant (cartes de lavage de voitures – à 8 contacts – des stations service BP, par exemple) ;
- programmer un identifiant librement choisi dans une carte vierge ou disposant d'une zone de mémoire encore disponible en écriture.

Dans une carte (épuisée) de 24 unités de lavage, par exemple, il reste normalement 136 bits à 0 qu'il est facile de mettre individuellement à 1 (voir notre ouvrage *Cartes à puce – Initiation et applications* dans cette même collection).

Chacune de ces façons de procéder correspond à un compromis différent entre sécurité et souplesse d'utilisation, et ne doit donc pas être adoptée à la légère.

La voie que nous allons explorer ici consiste à personnaliser le programme du PIC en fonction du contenu des cartes que l'on souhaite reconnaître.

La mise en œuvre est, certes, un peu plus lourde que par « apprentissage », mais la sécurité d'ensemble s'en trouve renforcée, tandis qu'un PIC moins puissant suffit amplement.

La simplicité du programme n'aura d'égale que celle du schéma de la **figure 1.9**, en dépit de la complexité (toute relative !) de la tâche à accomplir.

Ne consommant du courant (et d'ailleurs fort peu) que lorsqu'une carte est présente dans le connecteur, le montage peut fonctionner

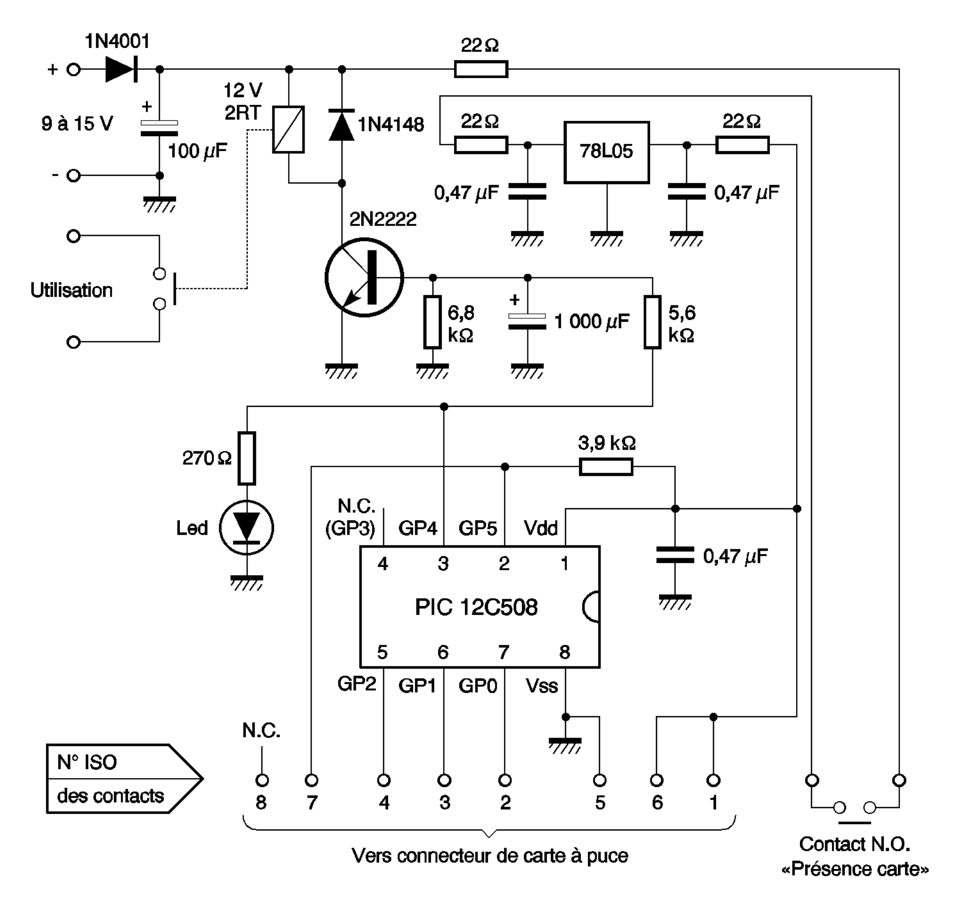


Figure 1.9.
Schéma de la serrure personnalisable.

sur une simple pile de 9 V, mais il n'est nullement interdit d'utiliser une alimentation secteur ou une batterie (par exemple celle d'une centrale d'alarme).

Dans tous les cas, un régulateur 78L05 fait travailler le PIC sous 5 V, tandis qu'un transistor permet au relais de fonctionner à partir de la tension non régulée (9 à 15 V).

La réalisation pratique du module principal suppose la gravure d'un circuit imprimé conforme au tracé de la **figure 1.10**, puis son câblage selon le plan de la **figure 1.11**.

Un bornier à quatre circuits sert au raccordement de la source d'alimentation et du circuit commandé par le contact du relais (utilisation), tandis qu'un tronçon de barrette sécable à double rangée de picots carrés coudés permet l'enfichage de la prise HE 10 à dix contacts du câble provenant d'un connecteur de cartes à puce.

Figure 1.10.
Tracé du circuit imprimé.

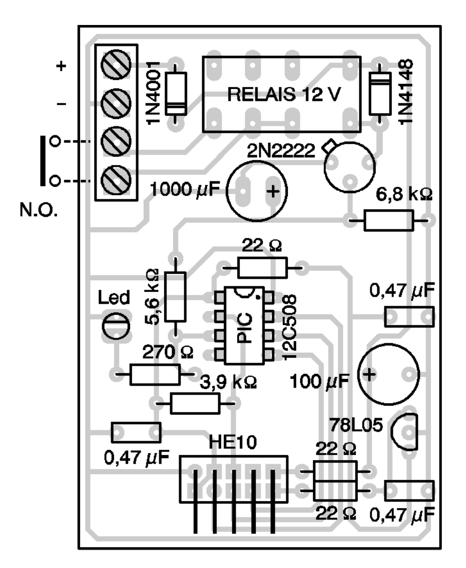


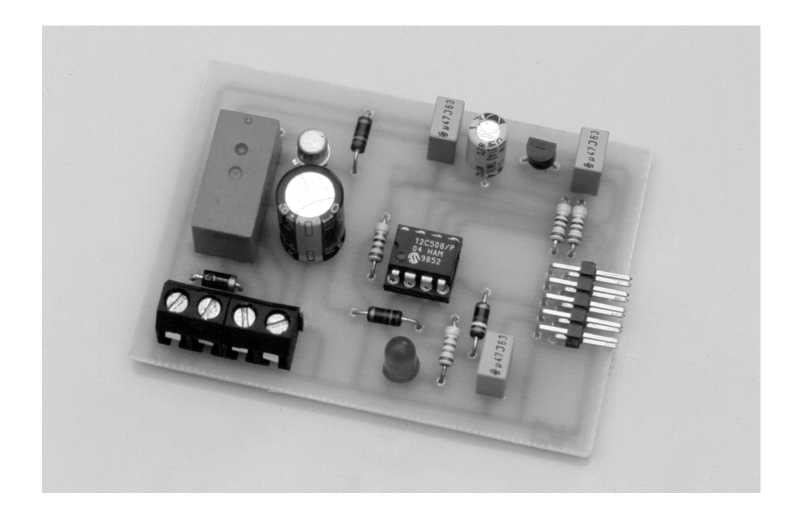
Figure 1.11.
Plan de câblage.

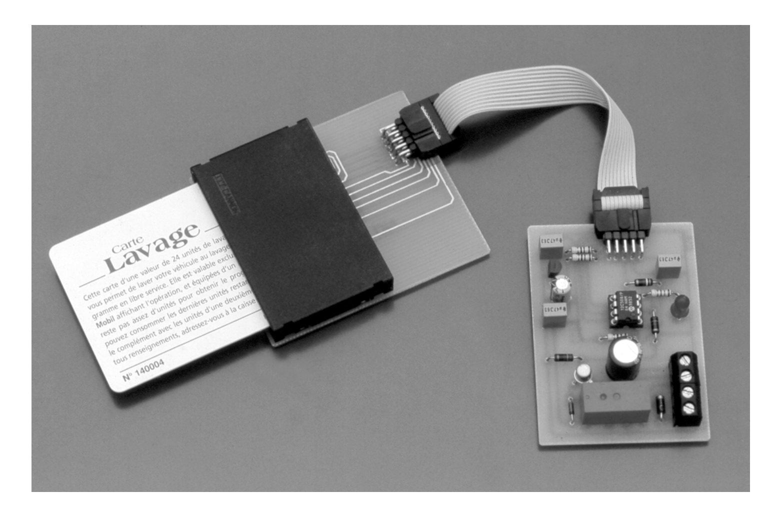
Liste des composants
1 PIC 12C508 à programmer (LOCK508.HEX ou TEST508.HEX)
1 × 78L05
1 × 2N 2222
1 × 1N 4001
1 × 1N 4148
1 Led rouge
3 × 22 Ω
1 × 5,6 kΩ
1 × 6,8 kΩ
1 × 3,9 kΩ
1 × 270 Ω
3 × 0,47 µF
1 × 100 µF 16 V radial
1 × 1 000 µF 6 V radial
Relais 12 V 2RT DIL
Bornier 4 circuits 5,08 mm
Barrette sécable à doubles picots carrés coudés

Le montage sera installé « en lieu sûr », autrement dit à l'intérieur des locaux protégés (par exemple au dos d'une porte), mais pas trop loin du connecteur (éviter de dépasser 50 cm de câble méplat).

L'exécution mécanique dépendra largement des conditions dans lesquelles sera utilisée la serrure, une technologie « antivandales » devant être envisagée dans les cas les plus difficiles.

Il aurait naturellement été fort imprudent de réunir tous les composants du montage sur le même circuit imprimé que le connec-





La serrure personnalisable.

teur de carte, faute commune à bien des développements « professionnels » dont il faut déplorer la conception aussi simpliste.

Dans certains hôtels, il suffit même de retirer les vis d'un (robuste !) plastron, pour pouvoir extraire tout le montage et accéder à deux fils qui, mis en court-circuit, ouvrent la porte de la chambre…

En tout état de cause, le brochage de l'embase est compatible avec les différentes variantes de connecteurs décrites dans nos précédents livres, et au chapitre 2.

On veillera évidemment à ce que le jeu de huit balais de contacts utilisé corresponde à la position (ISO ou AFNOR) de la puce des cartes employées !

Une fois encore, tout le secret de la simplicité de l'électronique réside dans le programme qu'exécute le PIC.

Comme le principe même de ce projet consiste à écrire et compiler un programme spécifique pour chaque application, nous nous bornerons à fournir deux exemples, néanmoins faciles à adapter à toutes les situations courantes.

```
DEVICE 12C508
DIM B0,B1,B2,B3,S
DEFINE PortB=%00101000
MOVLW 0 : OPTION
OUTB(0) : HIGH B.0 : HIGH B.1 : LOW B.1
LOW B.0 : HIGH B.1 : LOW B.1 : HIGH B.2
FOR B0=1 TO 5
GOSUB jump
NEXT B0
B3=117 : GOSUB check
B3=130 : GOSUB check
OUTB(16) : STOP
check:
B2=0
FOR B1=0 TO 7
B2=B2*2 : S=INPORTB & 32 : S=S/32: B2=B2+S
HIGH B.1 : LOW B.1
NEXT B1
IF B2 <> B3 THEN GOTO False
RETURN
jump: FOR B1=0 TO 7
HIGH B.1 : LOW B.1
NEXT B1 : RETURN
False: OUTB(16) : DELAYMS(255) : OUTB(0)
END
```

Le premier (TEST508.BAS) est écrit pour reconnaître indifféremment les « cartes lavage » (BP) de 12 ou 24 unités (à 8 contacts), et cela qu'elles soient neuves, épuisées, ou en cours d'utilisation.

Le programme se contente en effet de vérifier deux octets consécutifs (117d et 130d), qui se trouvent être communs aux identifiants des deux types de cartes, cela après avoir « sauté » les précédents sans même en prendre connaissance (sous-programme « jump »).

Bien entendu, cela ne se fait qu'après un « reset » en bonne et due forme de la carte, autrement dit après application d'une séquence bien précise et relativement complexe d'impulsions, garantissant que la lecture commencera juste au premier bit de l'espace mémoire.

Nos lecteurs familiers du langage machine des PIC remarqueront que la quatrième ligne est écrite en assembleur et non en BASIC. C'est une possibilité remarquable du compilateur LET Basic que de permettre de « mélanger » ainsi les deux langages sans précaution particulière…

En l'espèce, cela permet de modifier le contenu d'un registre spécifique du PIC (OPTION), dont la valeur par défaut ne nous convenait pas.

La compilation de ce code source produit un listing assembleur (TEST508.ASM), que son assemblage avec MPASM transforme en code machine (TEST508.HEX) prêt à être programmé dans le PIC.

On ne manquera pas de remarquer que le programme ainsi compilé n'occupe que 106 mots de mémoire sur les 512 disponibles dans le PIC, preuve tangible que ce compilateur génère un code redoutablement efficace, et par ricochet d'une foudroyante rapidité.

Écrire un programme plus complexe nécessiterait évidemment d'acquérir la version complète du compilateur, la version gratuite étant « bridée » en matière de nombre de lignes du code source.

Une remarque, enfin, au sujet de l'avant-dernière ligne du programme.

Lorsqu'une « mauvaise » carte est repérée, le programme allume la Led pendant 255 ms, ce qui alimente en même temps le transistor de commande du relais.

Comme celui-ci ne doit, bien évidemment, coller qu'en présence d'une « bonne » carte, un condensateur de 1 000 µF retarde suffisamment la conduction du transistor pour que le collage ne se produise pas.

Cet artifice permet de se contenter du fort petit nombre de lignes d'entrée-sortie disponibles sur le PIC 12C508, malgré la récupération de celles dévolues au circuit d'horloge (utilisation de l'oscillateur interne de 4 MHz).

Le second programme (LOCK508.BAS) réutilise la même séquence d'initialisation et de reset (les six premières lignes), puis teste cette fois les neuf premiers octets de la carte, c'est-à-dire 72 bits sur les 96 que compte la zone inaltérable.

```
DEVICE 12C508
DIM B1,B2,B3,S
DEFINE PortB=%00101000
MOVLW 0 : OPTION
OUTB(0) : HIGH B.0 : HIGH B.1 : LOW B.1
LOW B.0 : HIGH B.1 : LOW B.1 : HIGH B.2
B3=136 : GOSUB check
B3=128 : GOSUB check
B3=32 : GOSUB check
B3=2 : GOSUB check
B3=60 : GOSUB check
B3=117 : GOSUB check
B3=130 : GOSUB check
B3=36 : GOSUB check
B3=161 : GOSUB check
OUTB(16) : STOP
check:
B2=0
FOR B1=0 TO 7
B2=B2*2 : S=INPORTB & 32 : S=S/32: B2=B2+S
HIGH B.1 : LOW B.1
NEXT B1
IF B2 <> B3 THEN GOTO False
RETURN
False: OUTB(16) : DELAYMS(255) : OUTB(0)
END
```

Cela permet de ne reconnaître que les seules « cartes lavage » de 24 unités (à 8 contacts), qui sont toutes exactement identiques entre elles.

Appliqué à une télécarte, ce principe ne permettrait de reconnaître qu'une et une seule carte, sauf modification (très simple !) destinée à accepter, mettons, toutes les cartes de 120 unités en rejetant celles de 50.

Certains de nos lecteurs, vieux routiers du langage BASIC, s'étonneront peut-être de diverses maladresses ou lourdeurs apparentes de programmation.

En fait, celles-ci sont délibérées, et servent à contourner certains défauts de ce qui ne fut qu'une version provisoire du compilateur.

Les versions limitées postérieures à la 7.00 ne supportant plus que le PIC 16F84, il n'était en effet pas possible de profiter des améliorations imaginées entretemps (par exemple les instructions DATA, les parenthèses dans les formules, etc.) sans obliger nos lecteurs à acheter la version commerciale.



CHAPITRE PAGE

2 AUTOUR DES CARTES ASYNCHRONES

Avec l'équipement progressif des PC en lecteurs de cartes à puce certifiés PC/SC, on assiste petit à petit au ralliement des développeurs d'applications, professionnels ou amateurs, à ce standard vraisemblablement appelé à faire l'unanimité.

Cela d'autant plus que d'excellents outils de développement sont disponibles, dont les plus faciles à mettre en œuvre sont entièrement gratuits !

Il n'en faut pas davantage pour s'intéresser de très près à toutes sortes de cartes asynchrones existantes, et même pour produire ses propres cartes.

2.1 DÉVELOPPER EN PC/SC

L'initiative PC/SC n'aurait sans doute jamais vu le jour si Microsoft n'avait pas rêvé, excusez du peu, de s'approprier la carte à puce.

En fait, la grande aventure « Windows for smart cards » a tourné court, et ce sont des systèmes d'exploitation concurrents que l'on rencontre désormais dans les cartes.

Cela ne retire rien, tout au contraire, à l'intérêt de PC/SC, car il faut bien faire la distinction entre les deux domaines, certes complémentaires, que sont les applications destinées aux cartes, et celles qui pilotent les lecteurs.

Le développement d'applications embarquées dans les cartes est une spécialité à part entière, apparentée à de la programmation de microcontrôleurs, tandis qu'il devient courant d'intégrer des fonctionnalités « carte à puce » dans des logiciels destinés à « tourner » sur PC.

PC/SC est une opportunité exceptionnelle pour les développeurs d'applications Windows, qui peuvent désormais créer des logiciels supportant les cartes à puce, sans devoir enchaîner l'utilisateur final à tel ou tel modèle de lecteur bien spécifique, et pas forcément réutilisable pour d'autres usages.

Que ce soit d'origine ou par mise à niveau (installation des « Microsoft Smart Card Base Components »), il faut maintenant considérer que Windows intègre le support « natif » des cartes à puce et des lecteurs PC/SC.

Pour le développeur, cela se traduit, au plus bas niveau qui lui soit rendu accessible, par la possibilité (si ce n'est l'obligation !) d'utiliser les services d'une API baptisée WinScard.dll :

```
SCardAddReaderToGroupA
SCardAddReaderToGroupW
SCardBeginTransaction
SCardCancel
SCardConnectA
SCardConnectW
SCardControl
SCardDisconnect
SCardEndTransaction
SCardEstablishContext
SCardForgetCardTypeA
SCardForgetCardTypeW
SCardForgetReaderA
SCardForgetReaderGroupA
SCardForgetReaderGroupW
SCardForgetReaderW
SCardFreeMemory
SCardGetAttrib
SCardGetProviderIdA
SCardGetProviderIdW
SCardGetStatusChangeA
SCardGetStatusChangeW
SCardIntroduceCardTypeA
SCardIntroduceCardTypeW
SCardIntroduceReaderA
SCardIntroduceReaderGroupA
SCardIntroduceReaderGroupW
SCardIntroduceReaderW
SCardListCardsA
SCardListCardsW
SCardListInterfacesA
SCardListInterfacesW
SCardListReaderGroupsA
SCardListReaderGroupsW
SCardListReadersA
SCardListReadersW
SCardLocateCardsA
SCardLocateCardsW
SCardReconnect
SCardReleaseContext
SCardRemoveReaderFromGroupA
SCardRemoveReaderFromGroupW
SCardSetAttrib
SCardState
SCardStatusA
SCardStatusW
SCardTransmit
SCardRawPci
SCardT0Pci
SCardT1Pci
```

On devine que cela n'est pas aussi simple qu'on aurait pu le croire…

En pratique, on est fermement incité à acquérir le « Smart Card SDK » auprès de Microsoft, et à l'associer à un outil de développement Windows de même provenance…

Mais il existe fort heureusement d'excellentes alternatives !

Dans le répertoire « DELPHI » du dossier « PCSC » de notre cédérom, on découvrira ainsi comment ACS propose d'utiliser un environnement de programmation résolument concurrent.

Écrit en Pascal, WinScard.pas se charge ainsi de l'importation de tous les services de WinScard.dll, les rendant facilement accessibles depuis Delphi.

Il n'empêche qu'une manœuvre aussi élémentaire que l'affichage de la réponse au reset (ATR) de la carte présente dans un lecteur, nécessite un code source (PCSC_MCU_atr.pas) bien long par rapport à la simplicité de l'opération.

L'auteur n'appréciant guère la lourdeur, ne jurant que par le Basic, et préférant les outils de développement gratuits, il ne pouvait pas rester insensible aux charmes d'une tout autre approche, qu'il va maintenant vous présenter avec une certaine jubilation.

2.2 LA BASICCARD

Fondamentalement, la BasicCard est « la première carte à puce programmable en Basic », et même, depuis l'abandon de « Windows for smart cards » (dérivé de Visual Basic), « la seule carte à puce programmable en Basic ».

Il s'agit, en fait, d'une grande famille de cartes à « système d'exploitation ouvert », c'est-à-dire l'équivalent, pour les cartes à puce, de ce que fut le Basic Stamp pour les microcontrôleurs : une révolution !

En clair, chacun peut acheter, au détail et à des prix abordables, des cartes « blanches » pour y embarquer librement ses propres applications.

Cela rappelle furieusement les « Wafer cards » et leurs innombrables cousines, mais la similitude s'arrête là : même si la BasicCard est accessible au grand public (ce qui ne fait pas que des heureux…), c'est un produit résolument professionnel, largement conforme aux normes en vigueur.

Mais la grande originalité du système BasicCard, c'est qu'un même langage de programmation très simple (oui, le Basic !)

peut servir à programmer aussi bien des cartes, que des applications pour leur « terminal » (entendons par là un PC équipé d'un lecteur approprié).

Le kit logiciel étant entièrement gratuit (offert dans le dossier « BASICCARD » du cédérom de cet ouvrage, et téléchargeable sur http://www.basiccard.com), rien n'empêche de s'en servir pour développer, en quelques lignes de code source, des applications destinées à toutes sortes de cartes existantes.

Elles fonctionneront à merveille avec n'importe quel lecteur PC/SC !

On ne résistera cependant pas longtemps à l'envie de programmer aussi des cartes, pour toutes sortes d'usages, même des plus « pointus », et on ne regrettera assurément pas l'expérience.

Il sera alors avantageux d'acquérir un kit BasicCard « Pro » complet (http://www.hitechtools.com), qui rassemble un lecteur « CyberMouse », tous les logiciels de développement, un épais manuel « papier », et… un échantillonnage de cartes.

2.3 LE ZCBASIC

La BasicCard étant une création de la PME allemande ZeitControl, on comprend pourquoi son langage de programmation s'appelle ZCBasic.

L'auteur ne reviendra pas ici, une fois de plus, sur tous ses arguments en faveur du Basic : ses lecteurs pas encore tout à fait convaincus les trouveront dans son ouvrage *Basic pour microcontrôleurs et PC*, paru dans cette même collection.

Remarquons simplement que le ZCBasic est un Basic compilé (et non pas interprété comme GWBasic ou QBasic), de conception ultramoderne, qui peut même servir à développer tout autre chose que des applications « cartes à puce ».

À vrai dire, il faut le situer bien plus près de Visual Basic que du « Basic Microsoft » des ordinateurs familiaux d'il y a vingt ans.

Même s'il laisse de côté l'interface graphique de Windows (ce que nous nous plaisons à appeler ses « fioritures »), le fait même qu'il supporte les lecteurs PC/SC prouve qu'il s'agit bel et bien d'un outil de développement Windows 32 bits.

Simplement, il exploite son « mode console », c'est-à-dire un mode « texte » plutôt rustique, dans une « fenêtre MS-DOS » de Windows 95 ou supérieur.

Cela ressemble d'ailleurs à s'y méprendre à du MS-DOS, mais ce n'en est pas : un logiciel développé en ZCBasic V.3 ou V.4 ne s'exécutera pas sur un « vieux » PC fonctionnant sous DOS ou sous Windows 3.

Comme tout Basic compilé qui se respecte, le ZCBasic est tributaire d'un compilateur (ZCMBASIC.EXE).

C'est lui qui produira de véritables exécutables Windows 32 bits, à partir du code source écrit au moyen de n'importe quel éditeur de texte.

Il est même fourni un environnement de développement en bonne et due forme, doté de tout le confort de Windows (ZCPDE.EXE), à partir duquel il est normalement prévu d'appeler le compilateur, ainsi qu'un excellent simulateur.

Mais il est également possible d'opérer directement en mode « ligne de commande », selon le mode d'emploi condensé suivant :

```
ZC-Basic Compiler Version 4.32 (c) ZeitControl 2002
Usage: ZCMBASIC [param [param...]] input-file [param [param...]]
    input-file:  The ZC-Basic source file (.BAS)
         param: -CT              Compile for Terminal (default)
                -C1.1            Compile for Compact BasicCard
                                 ZC1.1
                -C3.x            Compile for Enhanced BasicCard
                                 ZC3.x
                -CFconfig-file   Compile for card in config-file
                                 (.ZCF)
                -Dsymbol[=val]   Define symbol (default val=1)
                -E[exe-file]     Win32 Console application (.EXE)
                -Ipath           Search path for #Include files
                -OD[debug-file]  Debug file (.DBG)
                -OE[error-file]  Error file (.ERR)
                -OI[image-file]  Image file (.IMG)
                -OL[list-file]   List file (.LST)
                -OM[map-file]    Map file (.MAP)
                -Sstack-size     Set size of P-Code stack
                -Sstate          Set card state =
                                 [L]oad/[P]ers/[T]est/[R]un
```

Prenons ainsi pour exemple le court programme que voici, dont le code source porte, dans le dossier « PCSC » de notre cédérom, le nom de GETHIST.BAS :

```
#Include CARDUTIL.DEF
#Include COMMERR.DEF
ComPort=101
Call WaitForCard()
ResetCard (P$) : Call CheckSW1SW2()
```

```
CLS:Print"Caract. Historiques de l'ATR : ";
FOR F=1 TO Len(P$)
M=ASC(MID$(P$,F,1))
M$=HEX$(M):IF Len(M$)=1 Then M$="0"+M$
PRINT M$;" ";
NEXT F
Print:Print:Call WaitForNoCard
```

En supposant le kit logiciel de la BasicCard convenablement installé, et ce code source copié dans le dossier C:\BasicCardPro, il suffira de taper, dans une fenêtre MS-DOS ouverte sur ce même répertoire, la commande `ZCMBasic -CT -E GETHIST.BAS` pour obtenir un fichier exécutable GETHIST.EXE.

La taille respectable de celui-ci (252 928 octets) trahit bien le fait qu'il s'agit d'une application Windows au grand complet…

Son but est d'afficher les « caractères historiques » de la réponse au reset (ATR) de la carte présente dans le lecteur PC/SC.

Sensiblement ce que fait PCSCatr.EXE, donc, mais à partir d'un code source de quelques lignes, et non plus de plusieurs pages !

Tout ce qu'il y a de compliqué et de fastidieux dans la gestion d'un lecteur PC/SC (ou même de plusieurs !) est en effet pris en charge, de façon entièrement « transparente », par le ZCBasic.

Qu'il nous soit permis d'insister ! C'est tout à fait exceptionnel, et absolument remarquable en termes de simplification du développement d'applications PC/SC : quelques minutes au lieu de plusieurs heures, quelques heures au lieu de plusieurs jours ou semaines…

Certes, cela suppose de renoncer aux possibilités graphiques de Windows, mais lorsque cela posera vraiment un problème, on pourra s'intéresser à l'API BasicCard (également fournie sur notre cédérom), qui permet de programmer en Visual Basic, en C, ou en C++ si on en a l'habitude.

Quelques mots seulement, car c'est très simple, sur l'organisation générale de cet échantillon de code source ZCBasic :

Les deux premières lignes provoquent l'inclusion de deux bibliothèques fournies avec le compilateur.

Indispensable, CARDUTIL.DEF apporte les primitives de base comme l'attente d'une carte (`WaitForCard`), ou de son retrait (`WaitForNoCard`).

COMMERR.DEF, pour sa part, permet à `CheckSW1SW2` d'afficher les éventuels messages d'erreur carte « en clair », et non plus sous la forme peu parlante de codes numériques.

La ligne `ComPort=101` joue un rôle important, dans la mesure où elle indique que le programme doit se servir du lecteur PC/SC numéro 1. On lui substituerait `ComPort=1` si l'on voulait faire fonctionner, en mode « propriétaire », un lecteur « CyberMouse » branché sur le port série COM1:.

Cette ligne peut éventuellement être omise, mais il faut alors déclarer le lecteur utilisé, dans la variable d'environnement ZCPORT de Windows (ajouter la ligne `SET ZCPORT=101` dans l'autoexec.bat, par exemple).

En fait, la tâche principale tient en une ligne, la cinquième : on y déclenche, comme il faut systématiquement le faire, le « reset » de la carte, puis on s'assure qu'il ne s'est pas produit d'erreur.

Les caractères historiques de son ATR peuvent alors être récupérés dans la chaîne P$, précédemment affectée à cet usage.

C'est aussi simple que cela, et tout le reste n'est que de la mise en page !

Bien entendu, des applications infiniment plus ambitieuses peuvent être développées en ZCBasic, et nous en témoignerons très largement au fil des chapitres qui suivent.

2.4 DES LOGICIELS D'INVESTIGATION

En présence d'une carte à puce dont on ignore les « petits secrets », le premier réflexe est en général de tenter de lui « tirer les vers du nez ».

Par simple curiosité, tout d'abord, mais peut-être aussi pour se faire une idée personnelle du degré de confiance que l'on peut lui accorder, pour peu qu'elle soit censée sécuriser une application un tant soit peu « sensible ».

Que l'on ne nous fasse cependant pas dire ce que nous n'avons pas écrit : la technologie des cartes à puce est aujourd'hui adulte, et on sait parfaitement produire des cartes « intrinsèquement sûres », qui ne révéleront jamais les données confidentielles qu'elles contiennent, pas même sous la torture.

Par contre, tous les (vrais) experts s'accordent pour déplorer la négligence chronique de certains émetteurs de cartes, ou développeurs d'applications, qui ne se servent pas (ou pas bien) des mécanismes sécuritaires mis à leur disposition.

Cela pour gagner du temps au nom du sacro-saint « time to market », ou pour faire des économies qu'ils risquent pourtant de payer un jour au prix fort.

Si l'on propose ainsi au consommateur de louer un coffre-fort dont la porte s'ouvrira dès qu'il aura le dos tourné, eh bien autant qu'il le sache !

Et comme sa banque ne s'en vantera évidemment pas, c'est bel et bien à lui qu'il appartient de procéder à sa propre évaluation sécuritaire.

Au passage, nous nous offrirons même le luxe de présenter des moyens de sécurisation qui, bien qu'étant à la portée de l'amateur, se révéleront parfois plus sûrs que ceux que mettent (ou non !) en œuvre les « professionnels ».

Mais voyons un peu par quel bout prendre une carte à puce dont on ignore à peu près tout.

Un analyseur de réponse au reset

En tout premier lieu, il est intéressant d'interpréter sa « réponse au reset » (ATR), c'est-à-dire cette suite d'octets qu'elle émet spontanément dès sa mise sous tension.

Les normes ISO 7816-3 et 7816-4 décrivant par le menu la signification de chacun de ses bits, il est possible (sinon facile !) de développer un logiciel capable d'afficher un compte rendu détaillé.

Plutôt que de « réinventer la roue », nous ne pouvons guère mieux faire que de recommander chaudement ANALYSER.EXE, aimablement offert par ACS (avec son code source en C) sur le cédérom de cet ouvrage.

Notons que cet utilitaire fonctionne sous DOS (ou dans une fenêtre MS-DOS de Windows). Exception qui confirme la règle, donc, il n'est pas compatible PC/SC, et devra par conséquent être utilisé avec un « CyberMouse » ou un ACR 20 « série ».

Voici un exemple de rapport produit par ce logiciel, en l'occurrence à partir d'une carte bancaire « B0' » :

```
ATR Analyser : Developed by Advanced Card Systems Ltd.
ATR : 3F 65 25 08 36 04 6C 90 00
TS : 3F = Inverse convention
T0 : 65 = ,TB1,TC1,,5 bytes historical bytes follow
TB1 : 25 = II=1, P1=5
TC1 : 08 = N=8
----  F=372* D=1* I=50* P=5* volts N=8 etu ---- ('*'=default)
T01 : 36 : The meaning of these historical bytes are proprietary
T02 : 04 :
T03 : 6C :
T04 : 90 :
T05 : 00 :
---- End of Report ----
```

Le plus important est de remarquer que cette carte à protocole « T = 0 » fonctionne en « convention inverse », et également qu'elle se contente des paramètres de communication « par défaut » utilisés pour la transmission de son ATR.

Certaines cartes, en effet, peuvent proposer au lecteur de « négocier » une vitesse plus élevée, ce qui explique certains phénomènes qui apparaissent parfois lorsque l'on tente d'observer la suite du dialogue.

Les « caractères historiques », pour leur part, ont ici une signification « propriétaire », c'est-à-dire définie par une spécification propre à l'émetteur de la carte.

En présence d'une carte dont les caractères historiques ont une signification normalisée, ANALYSER.EXE procéderait bien évidemment à leur interprétation d'après la norme ISO 7816-4.

Un « scanner » de commandes

Les lecteurs de nos précédents ouvrages sur les cartes à puce savent bien que celles-ci obéissent à des « commandes » identifiées par deux octets : CLA (classe ISO) et INS (code opératoire).

L'ensemble des combinaisons CLA-INS reconnues par une carte donnée constitue ainsi le « dictionnaire » des commandes utilisables.

À défaut d'un document technique (confidentiel ?) qui en fournirait le détail, il est parfaitement possible d'en dresser la liste par une recherche systématique, bien évidemment automatisée.

La mise en application de cette idée, sous ZCBasic, va (déjà !) nous permettre de montrer comment pousser ce langage très au-delà de ses possibilités initiales.

En fait, nous allons faire appel à un premier outil pour déterminer la ou les classes que reconnaît la carte, puis à un second pour chercher, dans une classe donnée, quels codes opératoires sont reconnus.

Tout le problème, c'est que dans un programme « terminal », chaque combinaison CLA-INS que l'on souhaite employer doit être explicitement déclarée : pas question d'inclure tout bonnement CLA ou INS dans une boucle FOR-NEXT !

Qu'à cela ne tienne : nous allons carrément faire construire un code source ZCBasic par un programme lui-même écrit en ZCBasic !

Dans le répertoire « CLASSINS » du dossier « PCSC » du cédérom, on trouvera ainsi un fichier CLASS.BAS, et le résultat de sa compilation : CLASS.EXE.

L'exécution de ce programme « intermédiaire » produit le fichier CLA.BAS, tellement compliqué que sa compilation en CLA.EXE prendra un bon moment.

Exécutons maintenant ce dernier programme, et nous verrons progressivement s'afficher toutes les classes ISO que reconnaît la carte insérée dans le lecteur (PC/SC, bien sûr !).

Il peut n'y en avoir qu'une, par exemple BCh pour une carte bancaire ou A0h pour une carte SIM de téléphone portable, ou bien plusieurs.

00h est d'ailleurs une valeur « par défaut » assez répandue.

Il est intéressant d'observer comment certaines cartes reconnaissent une ou plusieurs classes « secondaires », inutilisées pendant la phase de vie « normale » de la carte.

Il est bien tentant d'imaginer que les commandes employant cette classe ne serviraient qu'en usine, lors de la production des cartes.

Précisons tout de même que chaque classe possible est essayée avec un code opératoire INS égal à 00h. Cette valeur ne correspondant, la plupart du temps, à aucune commande valide, on n'a normalement pas à craindre de déclencher une « catastrophe » dans la carte.

On sait bien, cependant, que le « risque zéro » n'existe pas…

Les choses sont un peu plus compliquées en ce qui concerne les valeurs possibles de INS, car il va falloir opérer une recherche systématique pour chaque classe reconnue par la carte (et c'est là que l'on risque de découvrir des différences suggestives).

Le programme INST.EXE, obtenu par compilation de INST.BAS, commence donc par demander quelle classe il va devoir tester.

À partir de cette valeur, il va construire un fichier INS.BAS, qu'il faudra encore compiler en INS.EXE (l'exécutable fourni à titre d'exemple opère avec CLA = 00h).

Lors de son exécution, il s'affichera petit à petit la liste complète des codes opératoires INS que la carte reconnaît, dans la classe considérée.

Cette fois-ci, chaque code opératoire possible est essayé dans une commande affublée de paramètres aussi invraisemblables que possible : P1 et P2 = FFh, Le = FFh, toujours en vertu du « principe de précaution ».

À titre d'illustration, voici les résultats obtenus avec une carte bancaire (classe BCh), périmée depuis 2002, et donc non encore compatible « EMV » :

```
........0E.10........20........30........40........50........
........70........80.82.84.86.88............A0....A8....B0...
.....C0........D0......................
```

2.5 DES LOGICIELS GÉNÉRALISTES

Dans le dossier « PCSC », nous avons ajouté quelques logiciels utiles, aimablement offerts par les partenaires qui nous ont bien aidés dans la préparation de cet ouvrage : ACS et ZeitControl.

SCARD.EXE, ou un équivalent, doit absolument faire partie de la panoplie d'utilitaires de tout possesseur d'un lecteur PC/SC, quelle qu'en soit d'ailleurs la marque.

Il permet, en effet, de composer de toutes pièces des commandes ISO 7816 pour dialoguer « en direct » avec n'importe quelle carte asynchrone.

Cette petite application supportant aussi bien les cartes T = 1 que T = 0, on notera que dans ce second cas, il faudra indiquer une longueur de bloc de données seulement dans Lc si la commande est « entrante », ou bien uniquement dans Le si la commande est « sortante », l'octet de longueur inutilisé étant alors mis à 00h.

De son côté, PCSCINF.EXE est un outil de test, servant à faire l'inventaire des lecteurs PC/SC connus du système d'exploitation, qu'ils soient actuellement connectés ou qu'ils l'aient été un jour.

Un lecteur logiciellement installé mais physiquement déconnecté sera ainsi listé, mais signalé comme étant en défaut.

Pour notre part, nous avons souhaité montrer à nos lecteurs avec quelle simplicité le ZCBasic peut, lui aussi, interroger le système d'exploitation au sujet des lecteurs PC/SC répertoriés.

Un code source d'une seule et unique ligne suffit ainsi pour afficher le nom du lecteur PC/SC numéro 1 (qui sera d'ailleurs bien souvent le seul !) :

```
If PCSCCount>0 Then Print PCSCReader(1)
```

Il convient d'admirer cette étonnante simplicité, en songeant au nombre de lignes de code qu'il faudrait pour obtenir le même résultat en utilisant les services de WinScard.dll à partir d'un langage plus conventionnel…

Dans le fichier LIST.BAS, nous avons tout de même ajouté une ligne supplémentaire évitant que, dans certaines configurations de Windows, la fenêtre MS-DOS ne se referme avant que l'on n'ait eu le temps de prendre connaissance de ce qui s'est affiché.

PCSCATR.EXE, enfin, est l'exécutable de l'application Delphi PCSC_MCU_atr.pas, présentée au début de ce chapitre, et dont tout le code source se trouve dans le répertoire « DELPHI » du dossier « PCSC ».

2.6 DES EXPÉRIENCES D'AUTHENTIFICATION

Jusqu'à présent, nous n'avons guère utilisé le ZCBasic que pour développer des applications « terminal », à expérimenter avec des cartes à puce existantes.

Voici venu le moment de découvrir avec quelle facilité cet étonnant langage permet aussi, à tout un chacun, de créer ses propres cartes !

Cela, avec des fonctionnalités sécuritaires basées sur les meilleurs algorithmes cryptographiques du moment.

C'est dans le répertoire « AUTH » du dossier « PCSC » de notre cédérom, que l'on trouvera tous les fichiers permettant de passer immédiatement à la pratique.

Comme dans le cas d'un programme « terminal », le code source (DONGLE.BAS) doit aussi être compilé avec ZCMBASIC.EXE, mais cette fois en un « fichier image » portant l'extension « .IMG ».

Spécifique à une version bien précise de BasicCard, celui-ci sera ensuite téléchargé dans la carte, au moyen soit d'un lecteur PC/SC, soit d'un « CyberMouse ».

Supposons donc que nous souhaitions produire une carte à partir de DONGLE.BAS et d'une ZC 3.9. Il faudra tout d'abord taper `ZCMBasic -C3.9 -OI DONGLE.BAS` pour compiler un fichier DONGLE.IMG.

Un utilitaire spécifique, BCLOAD.EXE, servira ensuite à programmer physiquement la carte, selon le mode d'emploi résumé suivant :

```
BCLOAD BasicCard Loader Utility Version 4.30 (c) ZeitControl 2002
Usage: BCLOAD [param [param...]] image-file [param [param...]]
  image-file:  The image file to download to the BasicCard (.IMG)
  param: -D              Display commands on screen
         -E[error-file]  Error file (.ERR)
         -L[log-file]    I/O log file (.LOG)
         -Pcom-port      Use specified COM port
         -Sstate         Set card state = [L]oad / [T]est / [R]un
```

Une ZC 3.9 étant insérée dans un lecteur PC/SC, on taperait ainsi `BCLoad -D -P101 DONGLE.IMG` pour opérer le téléchargement, en suivant à l'écran les progrès de l'opération.

Précisons que, sauf ordre contraire, la carte restera par défaut en mode « TEST », et pourra donc être reprogrammée aussi souvent qu'on le souhaitera.

Si l'on préférait se servir d'un lecteur CyberMouse ou ACR 20/30 S en mode « propriétaire », on utiliserait alors le paramètre `-P1` (lecteur branché sur le port COM1:) ou `-P2` (port COM2:).

Rappelons cependant qu'a ce mode « ligne de commande », dont l'extrême simplicité justifie notre préférence, il peut être substitué un « environnement de développement » intégré sous Windows.

Directement disponible dans le « menu démarrer programmes » après installation du « kit BasicCard », il prend en charge la compilation, la simulation et le téléchargement, en quelques clics de souris.

L'authentification « interne »

En matière de protection de logiciels, le bon vieux « dongle » branché sur un port série ou parallèle pourrait bien avoir fait son temps !

La tendance est en effet de plus en plus aux « clefs » s'insérant dans un port USB, ou même carrément aux cartes à puce.

Le présent projet est entièrement compatible avec ces deux approches, puisque la BasicCard servant de « Sésame » peut être insérée dans n'importe quel lecteur PC/SC, ordinaire ou en forme de fiche USB (tel le « SIMTracker » d'ACS).

Le prix très raisonnable de la BasicCard « enhanced » (ZC 3.3 ou ZC 3.9) permettrait ainsi d'offrir une protection par clef matérielle



Le « SIMTracker » : un lecteur PC/SC dans une fiche USB !

à des logiciels bon marché, qui ne pourraient s'accommoder du coût élevé des « dongles » conventionnels.

Et ne perdons pas de vue la totale souplesse que procure le système d'exploitation « ouvert » de la BasicCard : quelques lignes de Basic suffisent, et ce n'est qu'un exemple, pour ajouter un « compteur » bloquant l'utilisation du logiciel après un nombre donné d'utilisations !

Mais l'actualité est là pour nous rappeler que seule une cryptographie forte peut vraiment assurer l'inviolabilité d'une application « carte »…

Même si cette vérité « dérange », il serait en effet irresponsable de ne pas admettre que n'importe qui peut aujourd'hui fabriquer, « sur un coin de table », des cartes à puce reproduisant assez fidèlement certaines fonctionnalités de telle ou telle application existante.

C'est tout simplement la rançon du succès des cartes à « système d'exploitation ouvert » telles que JavaCard ou Multos, même si le « ticket d'entrée » n'est pas précisément à la portée du premier venu.

Sous sa forme de kit librement accessible au grand public, la BasicCard fait donc un peu figure de « pavé dans la mare », allant délibérément à contre-courant de ce fragile principe de sélection par l'argent.

Le même raisonnement s'applique parfaitement aux algorithmes cryptographiques, dont il est maintenant reconnu que les mécanismes les plus intimes doivent être largement rendus publics.

Entretenir le mystère autour du fonctionnement d'un algorithme ne peut servir qu'à retarder la découverte de ses éventuelles faiblesses, avec les conséquences catastrophiques que l'on imagine si des applications qui en font usage ont été massivement déployées entre-temps.

Les meilleurs algorithmes actuels se sont donc imposés dans un contexte de totale transparence : leur adoption en tant que standards n'est intervenue qu'au terme d'une mise à l'épreuve impitoyable par la communauté cryptographique internationale.

Cela garantit au passage l'absence de toute « porte dérobée » introduite sur requête de quelque autorité indiscrète…

Les différentes versions de la BasicCard supportent chacune tout ou partie de ces grands classiques, potentiellement plus sûrs que tel ou tel mystérieux algorithme « propriétaire ».

Les ZC 3.3 et ZC 3.9, fournies dans les kits commercialisés, supportent « en dur » le DES (à 56 bits, désormais dépassé) et le triple DES (à 112 bits).

Par le biais de bibliothèques à inclure, le développeur dispose aussi du SHA-1 (*Secure Hash Algorithm*), de l'ECC (*Elliptic Curve Cryptography* à 161 bits), et de l'IDEA (algorithme symétrique à 128 bits breveté par Ascom Systec Ltd.)

Et la BasicCard « Professional » ZC 5.4 propose désormais l'AES (*Advanced Encryption Standard*, basé sur le tout récent algorithme belge « Rijndael »).

Pour notre part, nous allons montrer avec quelle simplicité on peut mettre en place un mécanisme « challenge-response » basé sur le SHA-1.

Le rôle de n'importe quel dongle est de ne permettre l'exécution d'un logiciel (ou encore l'accès à des données) que s'il prouve au préalable sa présence et son authenticité.

L'un des moyens d'authentification les plus sûrs repose sur un processus « question-réponse » ou « preuve sans connaissance », comparable à celui utilisé pour sécuriser les cartes SIM des téléphones portables.

Une même clef secrète résidant à la fois dans le dongle et dans le logiciel ayant besoin de l'authentifier, un moyen simple doit permettre de procéder à une comparaison sans transmission effective d'aucun secret.

Dans la pratique, le logiciel élabore une chaîne de caractères plus ou moins aléatoire, qu'il soumet au dongle.

Celui-ci exécute un calcul complexe, ayant pour opérandes la clef secrète et la chaîne aléatoire, puis en transmet le résultat (nullement secret) au logiciel.

De son côté, le logiciel exécute le même calcul avec les mêmes opérandes, et doit donc obtenir le même résultat.

Si tel est bien le cas, il peut considérer que le dongle est authentique, car ce dernier lui a prouvé qu'il contenait la bonne clef secrète, sans toutefois transmettre celle-ci.

Intercepter et réutiliser une réponse du dongle ne servirait à rien, puisque la probabilité est infime que la même chaîne aléatoire soit de nouveau émise par le logiciel (si toutefois celui-ci a bien été écrit en respectant cette discipline sécuritaire).

Enregistrer et analyser un grand nombre de couples question-réponse serait également inefficace, pour autant que l'algorithme

ait bien été conçu pour déjouer cette forme courante de « cryptanalyse ».

L'algorithme SHA-1 se prête bien à ce genre de travail, et bénéficie précisément d'une grande confiance de la part de l'ensemble de la communauté cryptographique internationale.

Il est d'ailleurs employé dans les plus sécurisés des « iButtons » de Dallas, les DS 1961S et DS 1963S, à vocation monétique.

Inutile, donc, d'y chercher (car elles auraient depuis longtemps été trouvées !) des failles comparables à celles mises en évidence, dès 1998, par des universitaires américains dans le très confidentiel « COMP128 », l'algorithme d'authentification implanté par défaut dans les cartes SIM des téléphones GSM.

Tout le détail du fonctionnement du SHA-1 est en effet public, puisque décrit dans le document FIPS 180-1 du *Federal Information Processing Standards* américain (http://www.itl.nist.gov/div897/pubs/fip180-1.htm).

En pratique, SHA-1 extrait un « condensé » (Hash) de 20 octets, de toute suite d'octets de longueur quelconque que l'on voudra bien lui soumettre.

Dans notre cas, il s'agira de la concaténation de la clef secrète et de la chaîne aléatoire, à concurrence de 256 octets, ce qui est amplement suffisant pour l'usage qui est le nôtre.

Bien que prévu pour être chargé dans une ZC 3.3 ou dans une ZC 3.9, qui sont des cartes à protocole T = 1, le programme « DONGLE.BAS » a été conçu pour pouvoir, éventuellement, être réutilisé en mode T = 0.



La BasicCard « Professional » est bi-protocole T = 0 ou T = 1.

C'est pour cette raison qu'il est fait appel à deux commandes distinctes (DONG et RESP), respectivement entrante et sortante.

En pur T = 1, il aurait été possible de combiner ces deux commandes en une seule, à la fois entrante est sortante.

Cette légère complication accentue la ressemblance avec la commande « RUN GSM ALGORITHM » des cartes SIM, auxquelles nous avons d'ailleurs emprunté la classe ISO A0h et les codes opératoires 88h et C0h.

Voyons donc comment est programmée la commande DONG :

```
Declare ATR="DONGLE"
#include Sha-1.def
Public P$ As String
Public KEY$="(c)2001 Patrick GUEULLE"

    Command &HA0 &H88 DONG(S$,Disable Le)
    SW1SW2=&H9F14
    E$=KEY$+S$
    P$=ShaHash(E$)
    End Command

Command &HA0 &HC0 RESP(Lc=0,S$)
S$=MID$(P$,P1P2+1,Le)
End Command
```

En fait, le code source d'un programme « carte » consiste surtout à définir, limitativement, le jeu de commandes que devra reconnaître la carte.

Chacune d'elles est identifiée par un octet CLA (ici A0h) et un octet INS (ici 88h).

Les données entrantes ou sortantes, véhiculées par la commande, passent très simplement par une chaîne (en l'occurrence S$), étant ici précisé (`Disable Le`) que la commande est entrante (la carte reçoit des données, mais n'en retourne pas).

La seconde ligne de la commande fixe la valeur des deux octets de compte rendu (SW1 et SW2) que la carte doit renvoyer, une valeur par défaut de 9000h étant utilisée en l'absence d'instructions contraires (ici 9F14h).

La troisième ligne exploite le bloc de données reçu par la carte, en le concaténant avec KEY$: une chaîne qui, ayant été préalablement déclarée « `public` », est accessible à partir de tout point du programme.

La quatrième ligne applique l'algorithme SHA-1 à la chaîne E$ ainsi construite, opération rendue possible par l'inclusion préalable de la bibliothèque `Sha-1.def`.

Remarquons que la variable de chaîne P$ ayant, elle aussi, été déclarée « `public` », son contenu pourra être exploité par n'importe quelle autre commande, en l'occurrence RESP (une commande « sortante », cette fois-ci, `Lc=0` indiquant qu'il n'y a pas de données entrantes).

Pourrait-on imaginer plus simple que ces douze lignes de ZCBasic, qui suffisent pour mettre en œuvre les fonctions de base de ce dongle ?

Aussi avons-nous un peu corsé les choses en imaginant une option.

Il suffirait de retirer les mots `REM` du fichier DONGLE.BAS pour mettre en service huit lignes supplémentaires, lesquelles fixent une limite de dix sollicitations du dongle (correspondant, par exemple, à la période d'essai gratuit d'un logiciel).

Les quatre dernières lignes (également facultatives) définissent une commande ISO (`A0 90 00 00 00`) permettant de remettre le compteur à zéro, autrement dit d'autoriser dix nouvelles exécutions.

Bien entendu, il serait possible de perfectionner cette commande en introduisant un mot de passe, mais ce n'est là qu'une suggestion que nous soumettons à l'imagination de nos lecteurs (c'est alors, pourquoi pas, la carte qui pourrait générer une chaîne aléatoire permettant de crypter le mot de passe).

Outre le code source DONGLE.BAS, le cédérom contient deux fichiers image obtenus par compilation avec la version 4 du kit BasicCard :

- DONG33.IMG destiné à être chargé dans une ZC 3.3 ;
- DONG39.IMG destiné à être chargé dans une ZC 3.9.

Nos lecteurs n'ayant pas encore installé (ou mis à jour) le kit, pourront néanmoins programmer des cartes avec l'utilitaire BCLOAD.EXE fourni à cet effet (ne pas oublier de préciser, sur la ligne de commande, les options correspondant au type de lecteur de cartes à puce utilisé, PC/SC ou CyberMouse).

Reste maintenant à se servir de la carte !

En principe, le programme « terminal » n'est autre que le logiciel que le dongle sert à protéger.

Il peut être développé soit en ZCBasic, soit dans un langage Windows 32 bits permettant l'usage des API BasicCard (mais c'est nettement plus compliqué !).

Écrit en ZCBasic, CHECK.BAS se borne à afficher le message « `DONGLE RECONNU !` » si le processus d'authentification réussit, ou bien « `EXPIRATION !` » au bout de dix utilisations d'un dongle dans lequel on aura activé la fonction de limitation.

On y reconnaîtra la même directive d'inclusion de la bibliothèque SHA-1.DEF, ainsi que le même processus de concaténation de la clef secrète et de la chaîne aléatoire.

La génération de celle-ci associe plusieurs processus pseudo-aléatoires, bien que notre application ne nécessite pas vraiment un hasard de « qualité cryptographique » : il suffit que la chaîne aléatoire soit différente d'une fois sur l'autre, et autant que possible imprévisible.

La « graine » (*seed*) nécessaire à la « culture » de chaînes aléatoires est ainsi obtenue en construisant le texte hexadécimal d'une valeur numérique fournie (sur 4 octets) par la fonction RND du Basic.

Bien entendu, nos lecteurs pourront remplacer la dernière ligne par leur propre application écrite en ZCBasic, d'ailleurs pas forcément orientée « cartes à puce » puisque le ZCBasic est aussi un fort honnête Basic généraliste.

Une autre approche pourrait consister à programmer ici une sous-fonction indispensable à l'exécution d'un logiciel développé, par exemple, en Delphi, et que celui-ci appellerait, dans une fenêtre visible ou non, par un « CreateProcess ».

Sur le plan purement sécuritaire, une remarque s'impose toutefois.

Volontairement composée d'un texte intelligible à la lecture, la chaîne servant de clef secrète (KEY$) apparaît en clair dans l'exécutable obtenu par compilation !

Une façon élégante d'éviter de ruiner ainsi toute la sécurité du système consiste à traiter CHECK.EXE avec un quelconque logiciel « compacteur d'exécutables » (voir, par exemple, http://www.blinkinc.com).

Outre la réduction de taille qui est habituellement le but principal de la manœuvre, il en résulte généralement un cryptage qui brouille complètement la clef.

Tel qu'il est compilé, ce programme nécessite un lecteur de cartes à puce compatible PC/SC, qui peut être relié indifféremment à un port série ou USB, à condition que les drivers correspondants soient correctement installés.

À défaut de lecteur PC/SC, il suffirait de remplacer la ligne `ComPort=101` par `ComPort=1` ou `ComPort=2` pour pouvoir utiliser la carte « dongle » dans un CyberMouse série branché sur COM1: ou sur COM2:.

Les heureux possesseurs d'un kit BasicCard du commerce devront donc recompiler CHECK.BAS ainsi modifié, s'ils désirent expérimenter cette possibilité, ou tout bonnement s'ils souhaitent mettre en service les lignes REM.

L'authentification « externe »

Nous venons de montrer comment transformer une BasicCard « Enhanced » en un « dongle » capable de prouver son authenticité au terminal dans lequel elle est insérée.

Réciproquement, une carte à puce peut avoir besoin de vérifier que l'entité qui cherche à communiquer avec elle est bien habilitée à le faire.

Cela peut s'effectuer au moyen d'un mécanisme encore plus simple, suffisamment même pour pouvoir être programmé dans une BasicCard « Compact ».

Dans le jargon des développeurs d'applications « cartes à puce », on appelle volontiers « authentification interne » le processus permettant à une carte de prouver son authenticité, sans pour autant révéler aucun secret.

Le « dongle » dont nous venons de décrire la réalisation fonctionne selon ce principe.

Dans le processus inverse, communément appelé « authentification externe », c'est le terminal qui doit prouver à la carte qu'il connaît la clef secrète qu'elle contient, mais qui ne doit être transmise sous aucun prétexte.

C'est alors la carte qui fournit, à la demande du terminal, un nombre aléatoire, et le terminal qui retourne à celle-ci le résultat d'un calcul cryptographique effectué, là encore, par les deux parties en présence.

Si l'authentification interne sert surtout à n'accorder une prestation (exécution d'un logiciel, accès à un réseau, validation d'un paiement), qu'au détenteur d'une carte bien précise, l'authentification externe a plutôt pour vocation de protéger l'accès à des données « sensibles » stockées dans la carte elle-même.

Dans un cas comme dans l'autre, on est à cent lieues de l'incroyable naïveté avec laquelle certaines applications « professionnelles » se contentent d'un compte rendu de « bonne exécution »

(9000h), voire de la simple absence de compte rendu d'erreur, pour admettre l'authenticité d'une carte à laquelle elles viennent de transmettre (en clair, s'il vous plaît…) un code réputé « confidentiel » !

Développé à partir d'une BasicCard « Enhanced » (dotée de 8 Ko d'EEPROM), notre « dongle » a pu mettre à contribution l'un des meilleurs algorithmes cryptographiques actuellement disponibles, le SHA-1.

Cela n'est pas possible avec la BasicCard « Compact » (ZC 1.1), pourvue de seulement 1 Ko d'EEPROM, et dont l'algorithme « SG-LFSR » est avant tout prévu pour crypter les échanges de commandes entre la carte et le terminal.

En revanche, la ZC 1.1 coûte très sensiblement moins cher, tandis qu'il n'est pas inintéressant de voir ce que l'on peut faire avec cette carte « d'entrée de gamme » dont un échantillon (si ce n'est deux) a longtemps été fourni dans les kits BasicCard.

Deux courts programmes ZCBasic suffisent pour se fixer rapidement les idées, quitte à « corser » un peu l'algorithme cryptographique lorsque viendra le moment de monter une véritable application pratique.

Le programme (AP.BAS) destiné à être implanté dans la BasicCard ZC 1.1 se compose essentiellement de quatre définitions de commandes, auxquelles nous avons arbitrairement affecté une « classe ISO » égale à 50h.

La commande « RAND » (code opératoire 02h) a pour unique objet de construire une chaîne aléatoire longue de quatre octets, soit 32 bits.

Rien n'interdirait de faire plus, mais cette valeur est cohérente avec les quatre à huit caractères des codes « PIN » qui protègent bien des applications dites « sensibles ».

La fonction RND du ZCBasic générant des nombres aléatoires de type « Long » (entiers signés de quatre octets, compris entre – 2147483648 et + 2147483647), un petit « bricolage » convertit son résultat en une chaîne de quatre caractères (DAT$).

La commande « RESP » (code opératoire 04h) exécute un très simple algorithme cryptographique à partir de la chaîne aléatoire produite par la dernière commande RND, et d'une clef secrète qui se compose, pour les besoins de la démonstration, des quatre octets ABh, CDh, EFh, AFh.

L'algorithme se borne à appliquer un opérateur logique AND, OR, ou XOR, à chacune des quatre paires d'octets extraites des

opérandes que sont la chaîne aléatoire et la clef, ce qui conduit bien évidemment à un résultat tenant, lui aussi, sur quatre octets.

Rien à voir, donc, avec une fonction de type « Hash », infiniment plus complexe, qui élaborerait plutôt un « condensé » des données entrantes.

La commande « TEST » (code opératoire 06h) utilise volontairement le même algorithme et la même clef, cette « faute de sécurité » étant tout simplement commise de façon à faciliter les expériences à venir.

Dans le cadre d'une véritable application pratique, il est bien évident qu'il faudrait imaginer une autre « formule secrète » !

Cela étant précisé, le but de cette commande est de débloquer les droits d'accès aux données que contient la carte, si et seulement si le bon « code confidentiel » est présenté par le terminal.

Ce code doit être recalculé à chaque fois, à partir d'une chaîne aléatoire demandée à la carte, et de la clef secrète qui doit naturellement être identique dans la carte et dans le programme « Terminal ».

La carte répond simplement par un compte rendu 98 04 si le code est erroné, et par 90 00 s'il est exact, auquel cas elle affecte la valeur 123 à un octet interne (FLAG) dont la valeur par défaut est 0.

Une variante présentant un niveau de sécurité supérieur pourrait être imaginée, selon laquelle il faudrait exécuter plusieurs fois de suite le couple de commandes « RAND » et « TEST », chaque succès incrémentant l'octet FLAG, et chaque échec le remettant à 0.

La commande « OUT » (code opératoire 08h), enfin, transmet au terminal un mot de sept octets (ici « Patrick ») si et seulement si la carte a accepté le code préalablement présenté (sinon, elle renvoie juste un compte rendu 98 08).

Bien entendu, ce simple exemple pourrait être étoffé, car il reste un peu plus de 300 octets d'EEPROM pour héberger d'autres données protégées, tandis que rien n'interdit de passer à une ZC 3.3 ou ZC 3.9.

La démonstration des possibilités de la carte ainsi « personnalisée » peut se faire à l'aide d'un très simple programme (AUTH.BAS) compilé pour le « terminal » (en général le PC équipé du lecteur de cartes à puce « CyberMouse » ou PC/SC ayant servi à programmer la carte).

On y retrouve naturellement les déclarations des quatre commandes définies dans le programme « carte », mais aussi le même algorithme et la même clef.

On pourra vérifier que la moindre modification effectuée d'un côté, mais pas de l'autre, fait échouer toute tentative d'authentification, et par conséquent d'accès aux données protégées.

Il est par contre possible, et même vivement conseillé, que chacun apporte des modifications (voire des améliorations !) de son cru à l'algorithme et à la clef, puis s'assure que tout fonctionne bien si celles-ci sont exactement identiques dans les programmes « carte » et « terminal ».

La vérification du bon fonctionnement de l'ensemble ne pourrait être plus simple : lancer AUTH.EXE (l'exécutable Windows obtenu par compilation de AUTH.BAS), sur un PC équipé d'un quelconque lecteur PC/SC.

Si tout va bien, l'insertion de la carte devra entraîner l'affichage du message « CARTE RECONNUE », puis du mot de données « Patrick ».

Éventuellement, on pourra détailler davantage les réactions de la carte en se servant d'un logiciel (tel que Scard.exe) permettant de lui envoyer directement des commandes ISO 7816 en protocole T = 1 :

- `50 02 00 00 04` pour lui demander quatre octets aléatoires (envoyer plusieurs fois de suite cette même commande, et constater que la réponse est chaque fois différente) ;

- `50 04 00 00 04` pour récupérer le résultat (xx xx xx xx) de l'exécution de l'algorithme à partir de la dernière valeur aléatoire générée, et de la clef secrète ;

- `50 06 00 00 04 xx xx xx xx` pour présenter le résultat précédent en tant que code confidentiel. Celui-ci sera reconnu comme exact si et seulement si les commandes RESP et TEST utilisent le même algorithme et la même clef (la fameuse « faute de sécurité » précédemment évoquée !)
  Bien entendu, il serait refusé si une nouvelle commande RAND était envoyée entretemps, sans toutefois être suivie d'une nouvelle commande RESP ;

- `50 08 00 00 07` pour tenter de lire le mot de données théoriquement « protégé ».
  On n'hésitera pas à envoyer cette dernière commande à différents stades de l'expérience, afin de vérifier dans quelles circonstances la carte accepte ou refuse de livrer les données qu'elle est censée protéger (et il n'est nullement interdit de risquer une comparaison avec certaines applications existantes !)

2.7 DES OUTILS CONNECTIQUES

Pour expérimenter commodément autour des cartes à puce, il est souhaitable de disposer d'un jeu d'accessoires facilitant au maximum les connexions que l'on peut être amené à établir entre les cartes et les « terminaux » qui les accueillent.

Adaptations de format et branchement en parallèle d'outils d'investigation ne sont, finalement, que deux exemples des bons et loyaux services que rendra volontiers un kit connectique spécialisé.

Un cordon « HE 10 »

Chacun sait qu'une carte à puce conforme à la norme ISO 7816 comporte au maximum huit contacts, dont deux ou trois restent d'ailleurs de plus en plus souvent inemployés.

Ajoutons à cela les deux connexions du contact de « présence carte » dont sont souvent équipés les connecteurs, et nous arrivons à un « devis » de dix conducteurs pour le cordon qui servira à interconnecter les éléments de notre « boîte à outils ».

Dès ses premiers pas dans le monde fascinant des cartes à puce, l'auteur de ces lignes a fait le choix d'un câble méplat, muni de deux fiches HE 10 à dix contacts, serties de façon à se trouver branchées en parallèle, fil à fil.

Ultérieurement, il est apparu nécessaire d'ajouter une troisième fiche du même modèle, vers le milieu du cordon, afin de pouvoir intercaler différents montages « espions » entre une carte et son lecteur.

Cela étant, la longueur idéale peut être fixée à 20 cm : plus court ne serait guère pratique, tandis que plus long ferait courir des risques de dégradation de la qualité des signaux.

Figure 2.1. **Le brochage du connecteur HE 10.**

La **figure 2.1** définit la correspondance des numéros ISO des contacts de la puce, avec les broches de l'embase HE 10 vue de face (côté insertion de la fiche), une encoche de détrompage éliminant toute ambiguïté.

Il est amusant de constater comment cette affectation, inspirée à l'origine du brochage d'un connecteur pour « clef à puce » avec lequel nous souhaitions assurer la compatibilité, a été systémati-

quement adoptée par nos confrères ayant rejoint, des années plus tard, le « club » des adeptes de la carte à puce…

Un connecteur multiformat

Même si toutes les cartes à puce « plein format » adhèrent depuis longtemps à la norme ISO pour le positionnement de leurs contacts, il ne faut pas oublier que des millions de cartes (et pas seulement de télécartes) ont été produites avec une « puce » en position AFNOR.

Beaucoup de connecteurs de cartes à puce sont d'ailleurs toujours équipés de seize balais de contact, même si la tendance semble évoluer vers des modèles à huit frotteurs (il n'y a pas de petites économies…).

Il n'est pas inutile de se doter des moyens de traiter les cartes à puce « AFNOR », ne serait-ce que pour pouvoir les essayer dans des lecteurs n'acceptant que les puces en position ISO.

C'est donc dans un souci d'universalité que nous avons dessiné le circuit imprimé (simple face) de la **figure 2.2**.

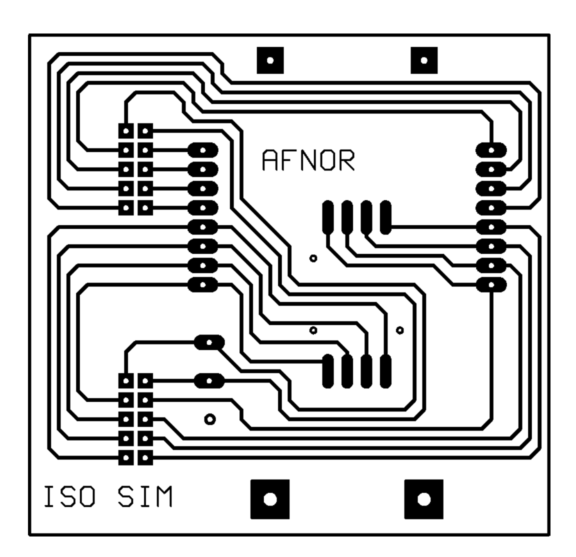


Figure 2.2.
Tracé du circuit imprimé porte-connecteur.

Côté composants (**figure 2.3**), il recevra un connecteur « ITT-CANNON » ou compatible, que l'on choisira équipé de seize balais et d'un contact « présence carte » de type « N.O. » (Normalement Ouvert, c'est-à-dire fermé lorsqu'une carte est présente).

Les deux embases HE 10 à dix contacts donnant accès, séparément, aux contacts ISO et AFNOR, pourront être remplacées par

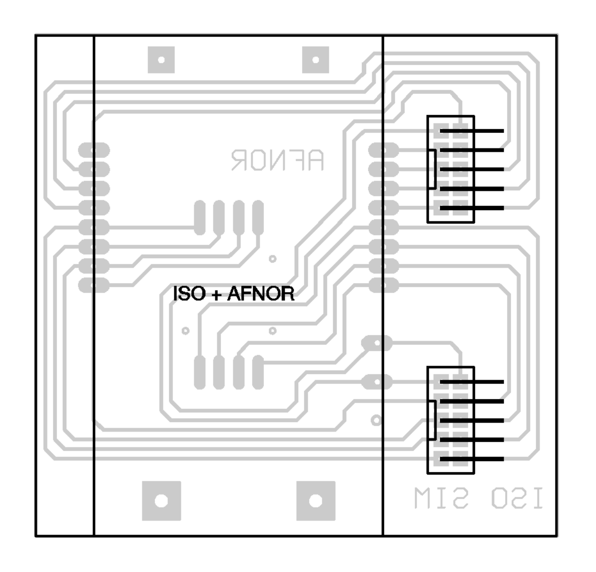
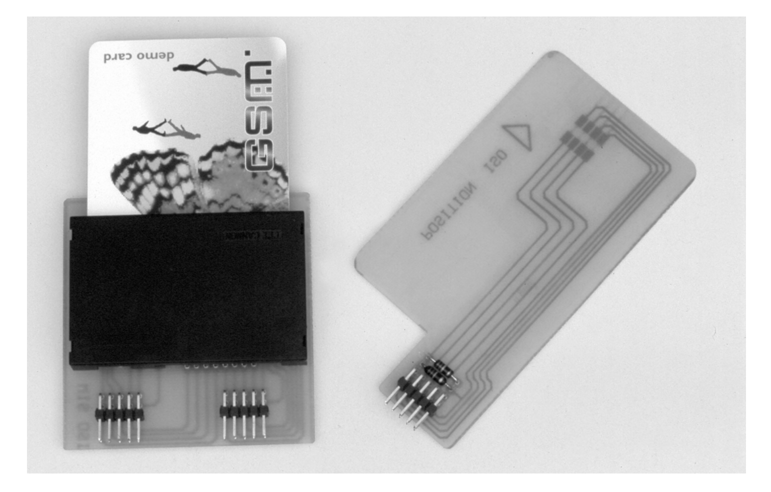


Figure 2.3.
Plan de câblage du côté composants.



Implantation du côté composants.

des tronçons de barrette sécable à double rangée de picots carrés coudés.

Ainsi, c'est le circuit imprimé lui-même qui fera office de détrompeur, l'ergot de la fiche venant buter dessus si celle-ci est présentée dans le mauvais sens.

Côté cuivre (**figure 2.4**), il est prévu de souder (dans le bon sens !) un connecteur pour cartes au format « SIM Micro », telles qu'en utilisent les téléphones portables.

Réalisé en technologie CMS, le modèle 33.7089 de Selectronic présente l'avantage d'être muni de huit contacts, contre seulement six pour la plupart de ses concurrents.

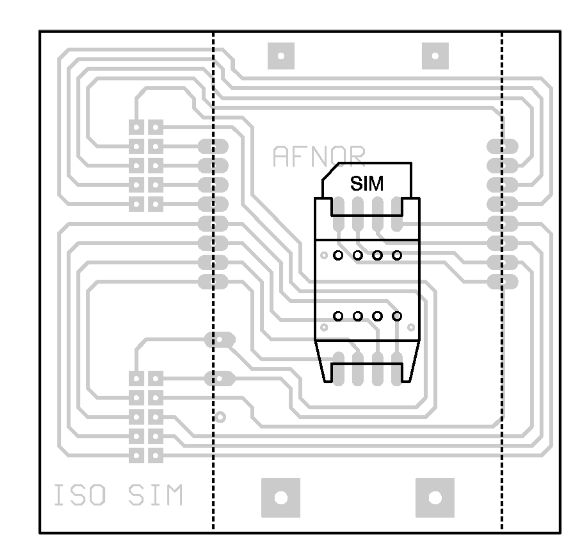
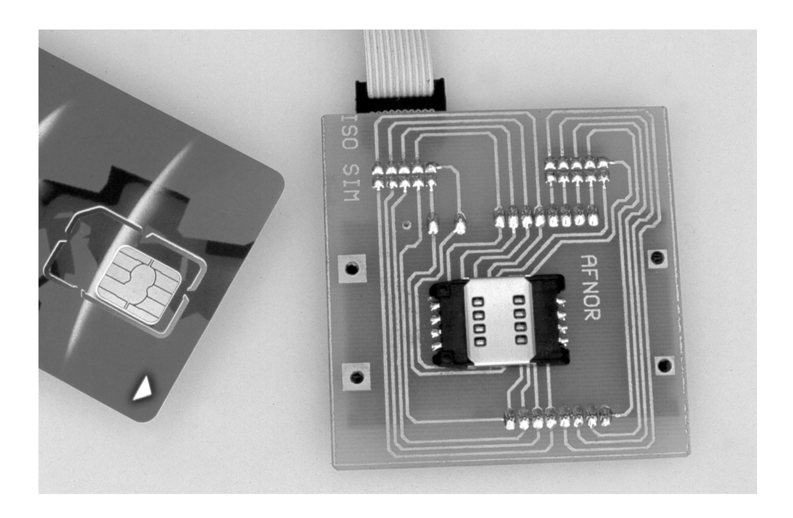


Figure 2.4.
Plan de câblage du côté cuivre.



Implantation du côté cuivre.

Cela permet, par exemple, d'utiliser des télécartes découpées à ce format très populaire, sans passer par un adaptateur mécanique.

Notons que le connecteur « SIM » étant branché en parallèle avec le jeu de balais « ISO » du connecteur « plein format », il convient de ne pas laisser une carte dans l'un lorsque l'on se sert de l'autre !

Une « fausse carte » ISO allongée

Sous le terme un rien provocateur de « fausses cartes », on regroupe en fait de multiples variantes de circuits imprimés d'épaisseur 8/10 mm, destinés à être insérés dans toutes sortes de « terminaux » en lieu et place de « vraies » cartes à puce.

Dotées d'une embase HE 10 au brochage conforme à la **figure 2.1**, celles-ci se prêtent à toutes sortes d'utilisations bien plus avouables que ce que pourrait laisser penser leur appellation.

Relier une « fausse carte » au connecteur que nous venons de décrire permet, par exemple, de faire accepter une carte AFNOR ou SIM Micro par un lecteur uniquement destiné aux cartes « plein format », avec puce en position ISO (CyberMouse, notamment).

Tel est aussi le cas de certains lecteurs de poche, capables d'afficher le solde d'unités d'une télécarte, mais seulement si sa puce est en position ISO.

Signalons enfin certains automates de lavage de voitures, qui refusent désormais les cartes prépayées avec puce en position AFNOR, dont la validité était pourtant illimitée !

Il a été longtemps d'usage de donner aux « fausses cartes » les dimensions exactes des vraies, ce qui ne posait aucun problème lorsque la moitié à peine de la carte était introduite dans le lecteur.

On rencontre toutefois de plus en plus de « terminaux », dans lesquels la carte pénètre en entier, sans pour autant être « avalée » par un système motorisé.

Cette évolution nous amène à allonger quelque peu la plaquette, afin qu'en toutes circonstances, l'embase HE 10 reste à l'extérieur du lecteur.

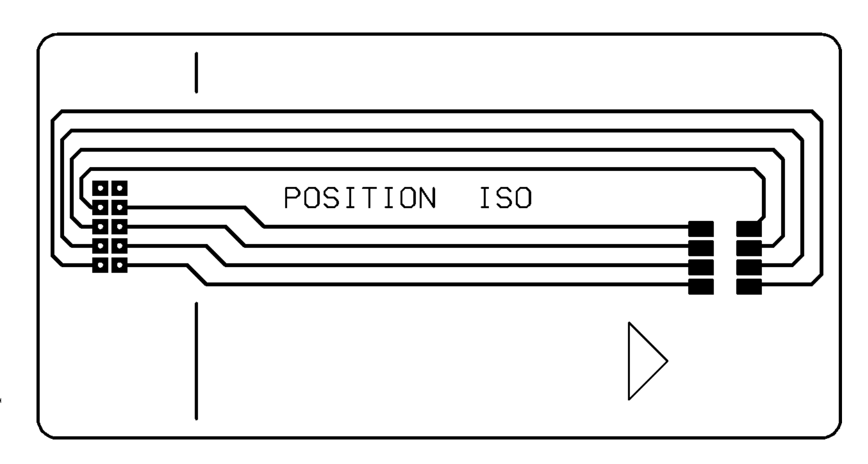


Figure 2.5. **Tracé du circuit imprimé de la « fausse carte ».**

Le tracé de la **figure 2.5** a été dessiné dans cette optique, sa longueur de 106 mm ayant été optimisée pour que la découpe d'une plaque présensibilisée de 100 × 160 mm dégage des chutes de dimensions encore utilisables.

Le cas échéant, la partie débordant du format « carte de crédit » pourrait être réduite à sa plus simple expression, par rognage à partir des amorces prévues à cet effet. Cela pourrait faciliter l'insertion dans certains lecteurs particulièrement « contrariants »…

Des « fausses cartes » SIM Micro

Le format « SIM Micro » est apparu avec la miniaturisation des téléphones portables, mais ce serait une erreur de penser qu'il se limite à ce domaine particulier.

Les « modules de sécurité » (SAM), qui équipent de plus en plus de lecteurs investis de responsabilités sécuritaires, l'épousent aussi très volontiers.

Et n'oublions pas toutes les applications rendues possibles par ces lecteurs de cartes « SIM Micro » qui se présentent sous la forme si commode d'une fiche USB !

Il est donc au moins aussi intéressant d'étudier le fonctionnement d'une carte SIM micro, que celui d'une carte de format ISO…

Pourtant, le remplacement de la « vraie » carte par une « fausse » est bien plus délicat dans un tel contexte de miniaturisation, d'autant que la carte est souvent complètement enfermée.

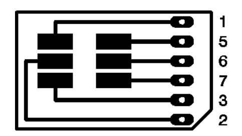


Figure 2.6.
Une « fausse carte » SIM Micro « minimum ».

Le minuscule circuit imprimé de la **figure 2.6** peut déjà rendre service dans un certain nombre de situations, car les six pastilles reliées à ses contacts se situent dans une zone souvent assez facile d'accès.

Il suffit alors d'y souder directement les six bons fils, parmi les dix d'un câble méplat terminé par l'habituelle fiche HE 10, pour pouvoir y connecter des accessoires externes.

Mais dans la plupart des cas, il faudra déployer des moyens plus élaborés !

C'est ainsi que la « fausse carte » de la **figure 2.7** pourra indifféremment être gravée sur du présensibilisé 8/10, que sur du circuit imprimé souple (Kapton) !

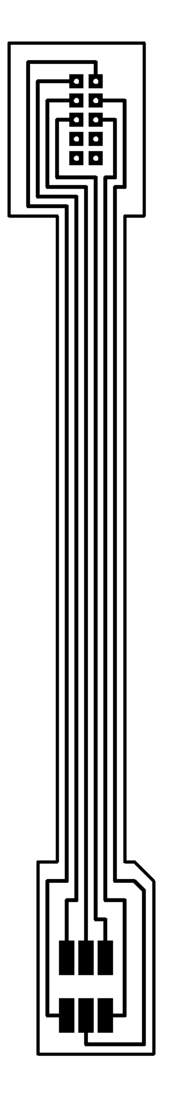


Figure 2.7.
Tracé de la « fausse carte » SIM Micro souple.

Il ne saurait évidemment être question de mettre directement en œuvre un film conducteur aussi fin, que l'on aura d'ailleurs tout intérêt à revêtir d'un plaquage d'or (kit de galvanoplastie disponible chez Conrad Electronique).

Du côté « carte », il suffira de le contre-coller (au ruban adhésif double face mince) sur une « fausse carte » découpée dans un quelconque matériau isolant suffisamment rigide, l'épaisseur totale devant se situer entre 0,75 et 0,8 mm.

Du côté HE 10 (**figure 2.8**), on exécutera un assemblage en « sandwich », entre deux plaquettes gravées, sur du stratifié 8/10, selon le tracé de la **figure 2.9**.

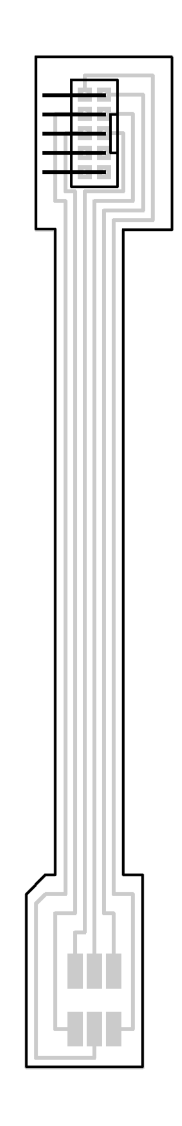


Figure 2.8.
Plan de câblage de la « fausse carte ».

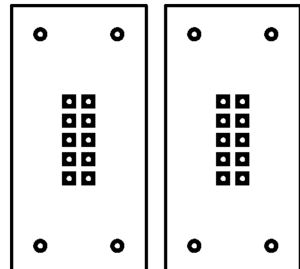


Figure 2.9.
Tracé des deux circuits imprimés raidisseurs.

L'une sera percée au diamètre de 1 mm (les trous des coins étant ensuite agrandis à 3 mm), avant d'y souder, mais cette fois-ci côté cuivre, l'habituel tronçon de barrette sécable à double rangée de picots carrés coudés.

La seconde sera percée à 2,5 mm, ce qui fera carrément disparaître les pastilles carrées : c'est voulu !

On poinçonnera alors les dix trous du circuit imprimé souple avec une pointe acérée (compas, par exemple), avant d'enfiler délicatement celui-ci (dans le bon sens, cuivre visible) sur ce qui dépasse encore des picots.

La continuité électrique sera établie soit par soudure (en cas de cuivre 35 µm sur Kapton), soit avec une résine conductrice (Elecolit 340 ou Jeltargent), si on a réussi à graver un autre film métallisé moins coûteux.

Il serait également possible de mettre en œuvre une crème à braser, grâce à un pistolet décapeur thermique.

Cette délicate opération achevée, on mettra en place la seconde plaquette (dont le rôle est purement mécanique), et l'on serrera le tout par deux boulons de 3 mm placés en diagonale.

Dans les cas où le tracé de la **figure 2.7** ne conviendrait pas, on pourra lui préférer celui de la **figure 2.10**, à câbler selon la **figure 2.11**.

On ne saurait d'ailleurs trop recommander, pour être paré à toute éventualité, de construire les deux versions !



Mise en œuvre des « fausses cartes » SIM Micro.

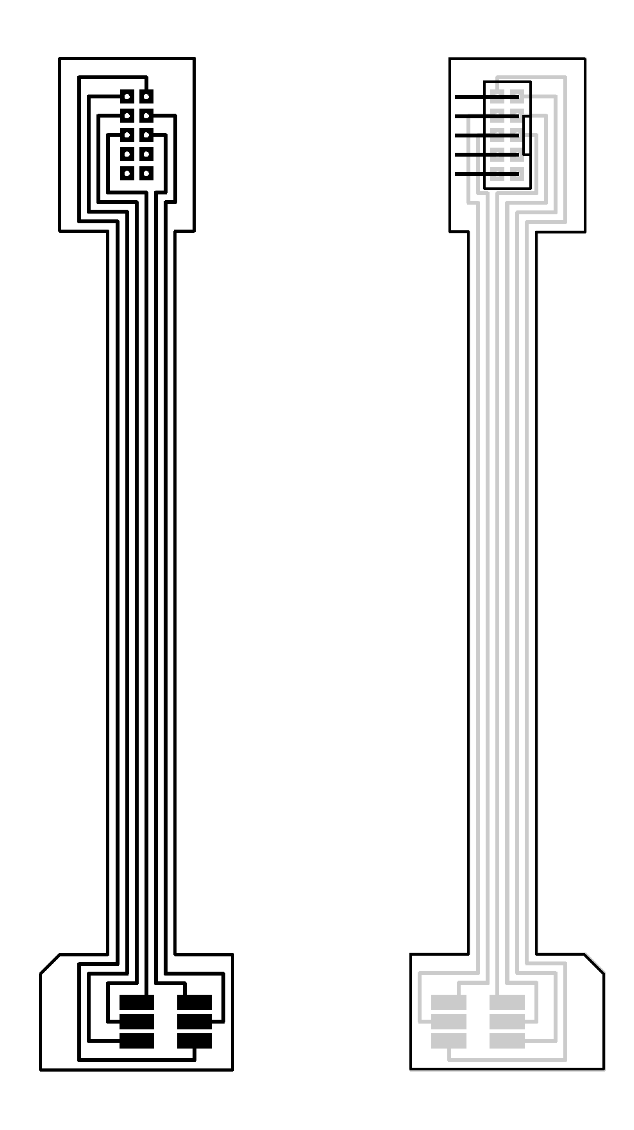


Figure 2.10. (à gauche) **Tracé de la « fausse carte » SIM Micro souple.**

Figure 2.11. (à droite) **Plan de câblage de la « fausse carte ».**

Un adaptateur pour fréquencemètre

L'une des applications les plus intéressantes de ce kit connectique sera, naturellement, le branchement d'un dispositif « espion » entre une carte à puce et l'appareil destiné à l'accueillir.

Ce pourra aussi bien être un téléphone portable (qui aura souvent énormément de choses à dire…), qu'un lecteur de cartes à puce de poche.

Dans un cas comme dans l'autre, il est toutefois courant que la fréquence d'horloge appliquée à la carte soit telle que le rythme de modulation diffère de l'habituel 9 600 bauds.

La dernière version de nos logiciels pour « espion » donnant la possibilité d'appliquer un facteur de correction *K*, il est utile de connaître la fréquence *F* réellement utilisée, avant de faire la « règle de trois » ($K = 12 \times 3{,}579/F$).

Certains lecteurs de poche, par exemple, produisent un signal d'horloge à 1 ou 1,5 MHz au lieu du classique 3,579 MHz, tout simplement pour limiter la consommation sur les piles : on appliquera alors, respectivement, un facteur de correction de 41 ou de 28.

Un petit accessoire fort simple suffit pour brancher, en toute sécurité, un fréquencemètre sur la liaison carte-terminal.

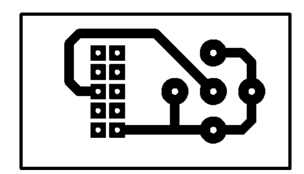
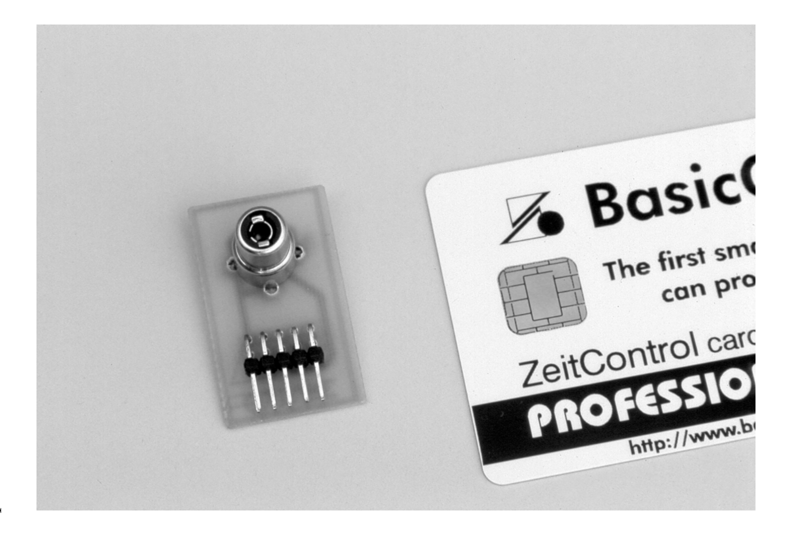


Figure 2.12. **Tracé du circuit imprimé.**

Gravé selon le tracé de la **figure 2.12**, un tout petit circuit imprimé sera équipé d'une embase RCA droite, et de la classique barrette sécable faisant office d'embase HE 10.



L'adaptateur pour fréquencemètre.

On branchera celle-ci sur la troisième fiche HE 10 du cordon reliant une « fausse carte » au connecteur de la **figure 2.4**, et un fréquencemètre numérique (ou un simple oscilloscope) sur la prise coaxiale.

C'est aussi simple que cela !

2.8 UN ESPION DE CARTES ASYNCHRONES

Sans revenir en détail sur un sujet abondamment traité dans nos ouvrages *PC et cartes à puce* et *Téléphones portables et PC*, il nous paraît utile de faire bénéficier nos lecteurs des dernières améliorations qu'il nous a été possible d'apporter à ce projet, dont se sont d'ailleurs largement inspirés certains de nos distingués confrères.

Rien de changé sur le plan matériel, comme en témoignent le tracé du circuit imprimé (**figure 2.13**) et le plan de câblage (**figure 2.14**), reproduits ici pour la commodité du lecteur.

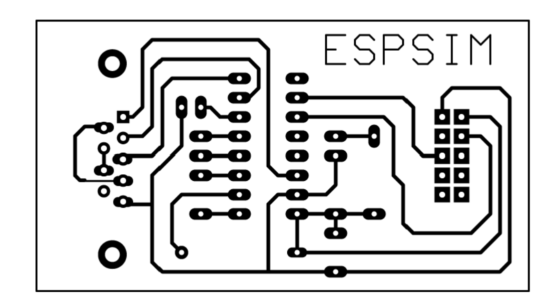


Figure 2.13. **Tracé du circuit imprimé.**

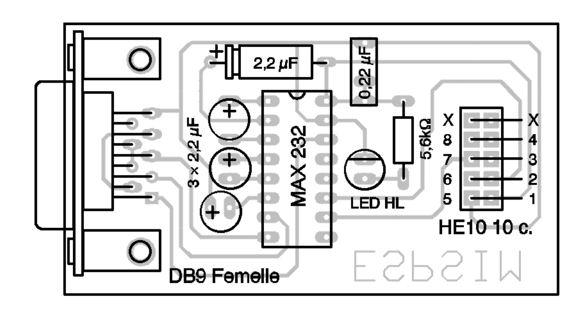


Figure 2.14. **Plan de câblage.**

Le cédérom du présent ouvrage contient par contre, dans son dossier « ESPASYNC », des logiciels sensiblement améliorés.

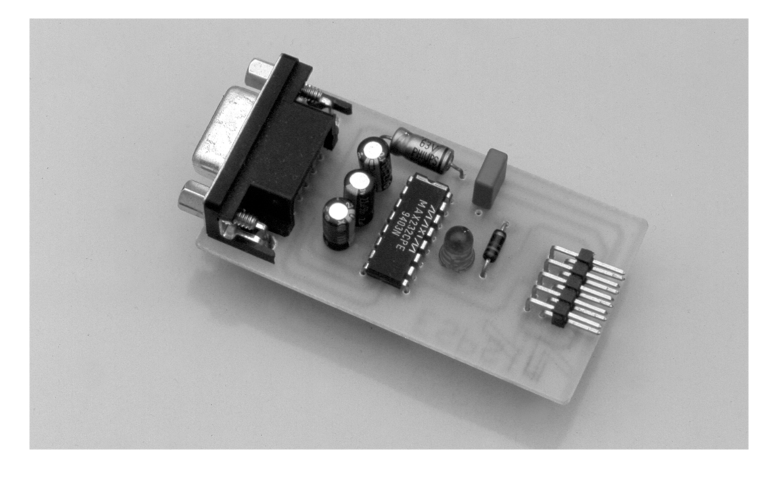
Cela, afin de tenir compte de l'évolution des caractéristiques des cartes à puce, de plus en plus difficiles à « espionner » sans recourir à de nouveaux artifices.

En pratique, ceux-ci permettent d'éviter les arrêts du programme, sur un message d'erreur, dès que le flot de données provenant de la carte diffère un peu trop de ce que veut bien décoder l'UART du PC.

Que dire, alors, sinon « bonne chance ! », à ceux qui ont la prétention de se contenter d'un « shareware » d'inspection de port série, tout bêtement trouvé sur Internet ?

Quatre variantes de notre programme sont fournies, tant en code source Turbo-Basic que sous une forme exécutable (MS-DOS), pour les cartes à convention directe ou inverse, et pour le port série COM1: ou COM2: du PC.

Bien entendu, le kit connectique dont nous venons de décrire la réalisation facilitera au maximum la mise en œuvre de ce puissant instrument d'investigation, dans toutes les situations susceptibles de se présenter.



L'espion de cartes à puce asynchrones.

Le cas le plus déroutant est celui des cartes, encore assez rares, qui appliquent la procédure dite « PPS » (*Protocol Parameters Selection*), en collaboration avec un lecteur compatible (comme les « Cyber-Mouse » et ACR 20/30).

Ces cartes se repèrent à certaines particularités de leur réponse au reset, que signale bien sûr tout bon logiciel d'analyse d'ATR.

Supposons que le paramètre *F* soit égal à 512 (au lieu de 372, valeur par défaut), et que *D* soit égal à 8 au lieu de 1.

Le rapport *F*/*D* passe ainsi de 372 à 64, ce qui signifie que le rythme de transmission, sur le contact ISO 7 de la carte, peut être près de six fois plus rapide que la valeur par défaut de 9 600 bauds ; cela, toutefois, sans pouvoir augmenter la fréquence d'horloge au-delà de 5 MHz (au lieu de 3,579).

Comme le lecteur n'est pas obligé de s'accommoder de ces paramètres, il peut entamer une « négociation » avec la carte dès que celle-ci aura fini de lui transmettre sa réponse au reset.

Ayant, par exemple, reçu une ATR égale à `3B 3C 94 00 4B 31 25 A2 10 13 14 47 83 83 90 00`, le lecteur pourra envoyer le bloc d'octets suivant à la carte : `FF 10 94 7B`.

En l’occurrence, cela signifie que le lecteur accepte de travailler à $F = 512$ et $D = 8$, mais il pourrait fort bien « marchander » en proposant ses propres paramètres.

La carte retourne alors exactement le même bloc d’octets, si elle est d’accord, mais elle peut aussi faire une contre-proposition, ce qui se traduit par la modification de quelques bits dans le bloc d’octets renvoyé.

À partir du moment où une telle négociation aboutit, l’espion n’affichera plus que des données erronées, tant qu’il restera réglé sur le débit de données par défaut.

CHAPITRE PAGE

3 AUTOUR DES CARTES SYNCHRONES

C'est avec des cartes synchrones, et en particulier avec des télécartes épuisées, que l'on commence généralement à « bricoler » les cartes à puce ; cela, en attendant de succomber à la tentation de s'attaquer aux cartes asynchrones (carte bancaire, carte Vitale, cartes SIM des téléphones portables, porte-monnaie électronique, etc.)

À ce stade, le savoir-faire progressivement acquis incitera bien souvent l'explorateur à s'intéresser de nouveau aux cartes synchrones, mais avec des moyens d'investigation nettement plus puissants.

Tel est précisément le thème de ce chapitre !

3.1 DES OUTILS D'EXPLORATION AVANCÉE

Dans leur grande majorité, les heureux possesseurs d'un lecteur de cartes à puce ACR 20, ACR 30, ou « CyberMouse » sont loin de se douter qu'il permet de lire et écrire dans toutes sortes de cartes à puce synchrones ; cela grâce à un logiciel fourni gratuitement par son fabricant (ACS), ou même en développant carrément une application « sur mesures » autour d'un contrôle ActiveX de même provenance.

À la fois performant et bon marché, le « CyberMouse » est largement compatible avec l'ACR 20 S (parfois baptisé « Yuhina »), et équipe notamment le fameux kit BasicCard.

Cela pour sa version série RS232, car il existe aussi un modèle USB (ACR 20 U), ainsi que des variantes « allégées », ACR 30 S et ACR 30 U.

Moyennant l'installation d'un driver approprié, tous ces lecteurs fonctionnent fort bien en mode PC/SC, ce qui permet de les considérer comme des lecteurs de cartes à puce « génériques » pour Windows (voir chapitre 1).

Dans cette configuration, on mettra essentiellement en œuvre des cartes à puce « asynchrones », autrement dit à microprocesseur, opérant en protocole T = 0 ou T = 1.

En mode « propriétaire », il est possible de développer des applications Windows dédiées, qui ne fonctionneront bien évidemment qu'avec un lecteur de la marque, cela par l'entremise d'un driver spécifique (ACSR2032.DLL).

Ce contexte se prête tout particulièrement à la découverte des vastes possibilités du CyberMouse en matière de cartes à puce synchrones.

Sans même écrire une seule ligne de code, il est possible de se livrer à d'innombrables expériences en installant simplement l'application « CardEasy », offerte par ACS sur le cédérom de cet ouvrage.

Son installation (Setup) met automatiquement en place le driver propriétaire, totalement indépendant du driver PC/SC : il faudra donc parfois « forcer » l'environnement dans lequel on souhaite travailler !

Si le driver PC/SC a déjà été installé, il ne faudra brancher le CyberMouse sur le port série qu'après le démarrage réussi de Windows (par exemple à l'aide d'un commutateur RS232 manuel) si l'on préfère opérer en mode « propriétaire ».

À défaut de cette précaution, le lecteur ne pourrait pas être détecté sur le port série, celui-ci étant déjà accaparé par le driver PC/SC.

L'application « CardEasy » doit être un peu considérée comme une « vitrine » des possibilités du CyberMouse, que le fichier d'aide et le manuel PDF décrivent par le menu (répertoire « DOC » du dossier « ACS » du cédérom).

En plus des cartes asynchrones, avec leurs classiques commandes ISO 7816 ou « APDU », le CyberMouse (ACR 20) supporte la quasi-totalité des cartes synchrones existantes, avec leurs éventuelles fonctionnalités spécifiques (par exemple l'authentification cryptographique des « Eurochip » !)

En pratique, le logiciel dialogue avec le CyberMouse au moyen de commandes spéciales, que le microcontrôleur du lecteur traduit en « micro-instructions » de bas niveau, propres à chacune des familles de cartes supportées.

Des restrictions frappent cependant les cartes nécessitant une tension de programmation *Vpp* (supérieure à 5 V), à commencer par les GPM 416 (que l'on pourra tout de même lire), et les télécartes « T1G » (pas directement acceptées par le CyberMouse, mais nous avons bien évidemment imaginé une solution à ce problème !)

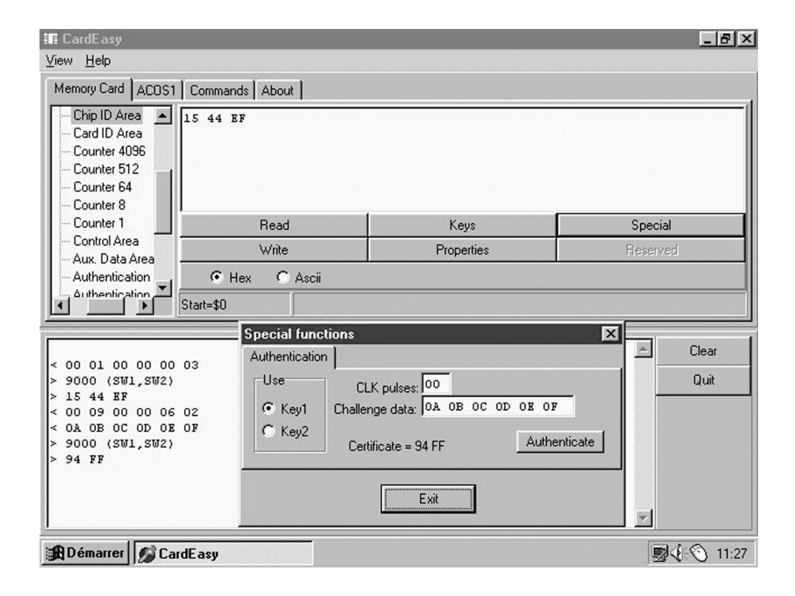
À ces exceptions près, chacun pourra librement opérer sur les familles de cartes suivantes, identifiées par un code tenant sur un octet :

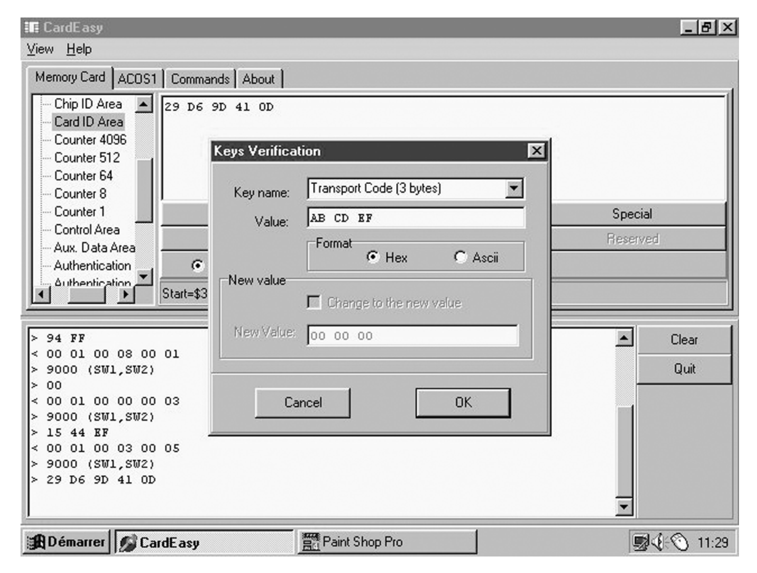
Cartes à protocole « Eurochip » (code famille 01h) :

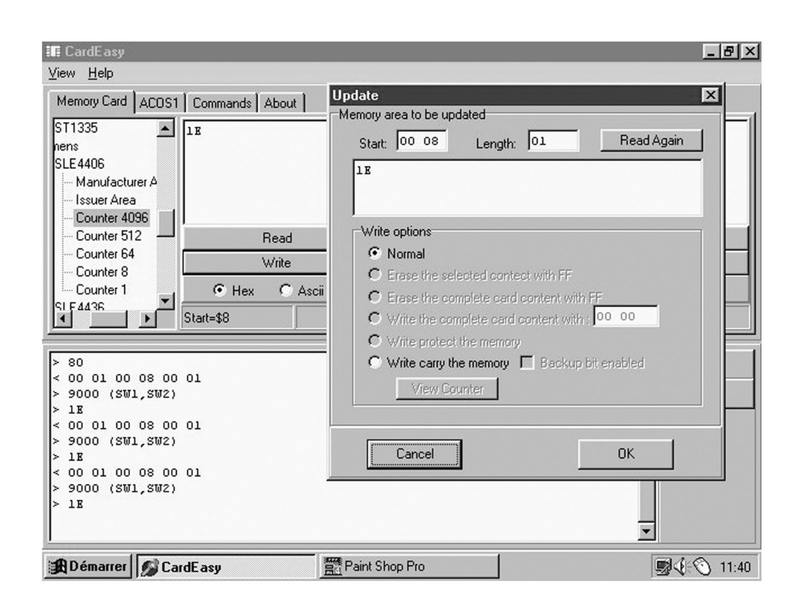
- *SLE4406, SLE443x, SLE553x (Siemens/Infineon)*
- *AT88SC06 (Atmel)*
- *ST1304, ST1305, ST1335, ST1336, ST1355 (STMicroelectronics)*

- *PCF2006, PCF203x, PCF223x (Philips)*
- *SCS152 (Microchip)*
- *GPM103, GAM226, GAM326 (Gemplus)*

Depuis la toute première « Telefonkarte » allemande jusqu'aux « Eurochip » de seconde génération, toutes les cartes de cette famille à protocole « 5 contacts », dont la capacité va de 104 à 271 bits, ont en commun un système de comptage d'unités de







type « boulier » (voir nos ouvrages *Cartes à puce – Initiation et applications* et *PC et cartes à puce*).

Chacun des compteurs à huit bits qui les équipent est affecté d'un « poids » huit fois supérieur à celui du compteur qui le suit dans l'espace mémoire.

Chaque unité consommée est matérialisée par la mise à zéro (écriture) d'un bit qui était encore à un dans le compteur « par un ».

Lorsque les huit bits de celui-ci se retrouvent à zéro, la mise à zéro d'un seul bit dans le compteur de poids immédiatement supérieur (donc « par huit ») autorise la remise à un (effacement) en bloc de tous ses bits, et ainsi de suite (écriture dite « avec retenue »).

Dans les cartes les plus simples de cette famille, quarante bits sont répartis en cinq compteurs (par 1, par 8, par 64, par 512, et par 4 096), ce qui permet déjà de compter jusqu'à plus de trente mille unités.

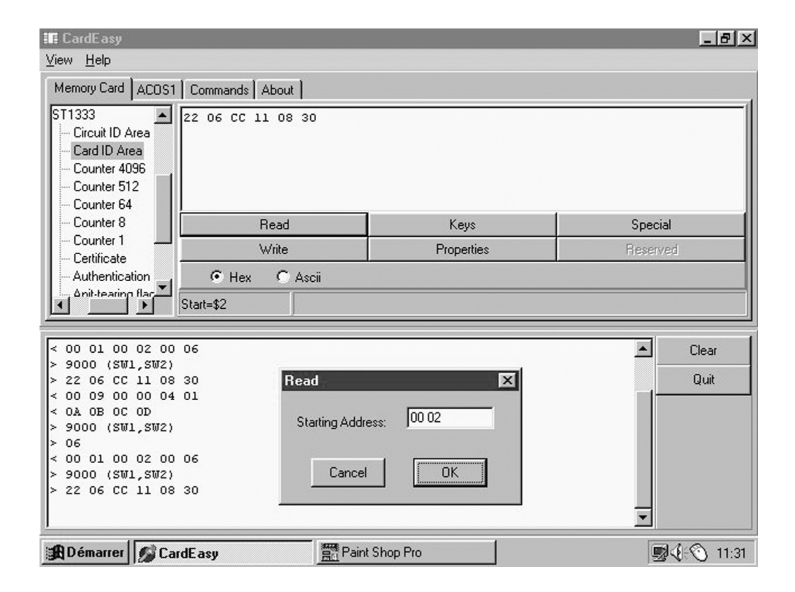
Parmi les variantes disponibles, on peut citer un compteur supplémentaire, des zones « utilisateur », reprogrammables après effacement, des bits de sauvegarde dits « anti-arrachement », et surtout des mécanismes d'authentification cryptographiques plus ou moins perfectionnés.

Dans tous les cas, une zone « fabricant » contient l'identification, inaltérable, du composant électronique interne, tandis qu'une zone « émetteur » peut être protégée contre toute modification une fois la carte mise en circulation.

Les cartes de cette famille sont principalement utilisées dans les publiphones des (nombreux) pays n'ayant pas adopté la technologie française, mais aussi dans diverses applications de type « carte à jetons » (petits paiements, fidélité, etc.)

Cartes à protocole « T2G » (code famille 0Bh) :

- *ST1303, ST1331, ST1332, ST1333, ST1353 (STMicroelectronics)*
- *TMS3582 (Texas Instruments)*
- *GPM271, GAM273 (Gemplus)*



La « télécarte de seconde génération » ou « T2G » a été imaginée dès 1989, sous le prétexte officiel de faire face à l'obsolescence de la technologie EPROM NMOS des T1G.

Il se pourrait bien, cependant, que la véritable motivation soit la lutte contre la fraude par « clonage », dont l'ampleur commençait sans doute déjà à devenir préoccupante.

Fruit d'un partenariat avec STMicroelectronics, la T2G fait appel au même principe de compteur d'unités à « boulier » que les cartes de la famille « Eurochip ».

Simplement, elle fonctionne selon un protocole de communication à 6 contacts au lieu de 5.

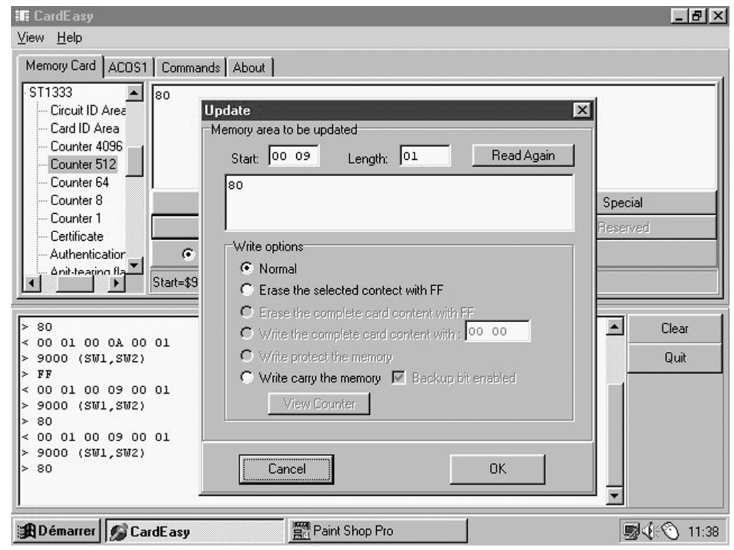
Contrairement à l'Eurochip, elle est également « vierge à 0 », ce qui signifie que l'on y « grille » des unités en mettant à un des bits qui étaient encore à zéro.

Parallèlement aux composants ST1303 ou ST1332, à l'usage exclusif de France Telecom, des versions dérivées ou même

concurrentes sont commercialisées pour les applications les plus diverses.

Certaines bénéficient d'un mécanisme d'authentification cryptographique (mais avec un algorithme différent !), et d'autres pas.

Bien des cartes de fidélité sont basées sur la version « non crypto » (ST1331 ou GPM271), moins coûteuse et plus simple à mettre en œuvre.



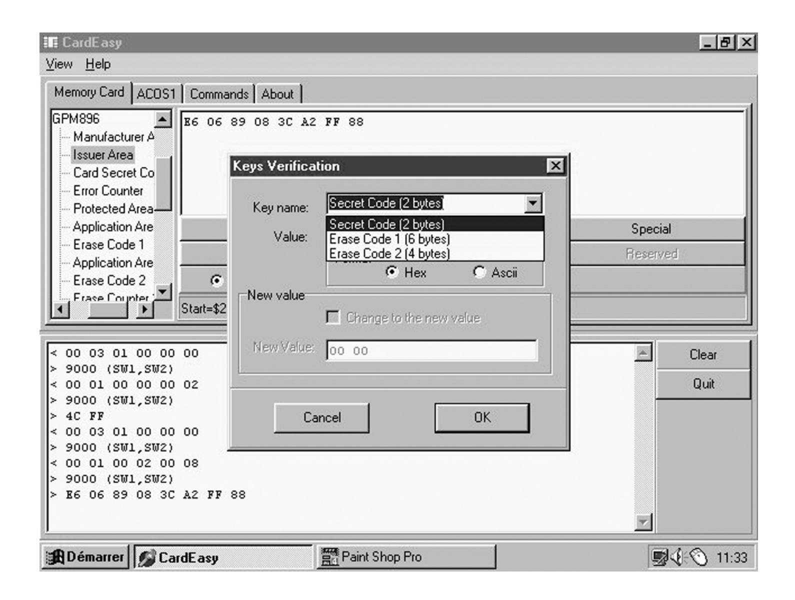
Cartes compatibles GPM416 (lecture seulement) / GPM896 (code famille 03h) :

- *SLE4404 (Siemens/Infineon)*
- *AT88SC101, AT88SC102, AT88SC1601, AT88SC1604 (Atmel)*
- *ST1301 (STMIcroelectronics)*
- *PCF7940 (Philips)*
- *GPM416, GPM896 (Gemplus)*

Équipée d'un composant ST1301, la GPM416 de Gemplus a marqué la transition technologique de l'EPROM à l'EEPROM.

Cette carte de « seconde génération » nécessitait toutefois encore une tension de programmation *Vpp* de 21 V, que le CyberMouse n'est pas en mesure de fournir (car il ne contient pas de convertisseur élévateur de tension).

Munie de zones protégées par des codes confidentiels, cette carte d'une capacité de 416 bits doit désormais être considérée comme obsolète.



Elle a cependant brillamment contribué au développement d'applications relativement ambitieuses, pour lesquelles des cartes asynchrones auraient été trop coûteuses.

La relève est heureusement assurée par la GPM896, carte de « troisième génération » d'une capacité accrue (896 bits), et intégralement compatible avec le CyberMouse, puisqu'elle se contente d'une tension unique de 5 V.

Notons d'ailleurs que le SLE4404, le composant concurrent du ST1301 chez Infineon, n'exige lui aussi que du 5 V.

On retrouve la GPM896 et ses équivalents dans des applications de type « cartes de fidélité », et de petite monétique (stationnement, clubs, restauration, transports, etc.)

Cartes à protocole « 3 fils » (code famille 05h) :

- *SLE4418, SLE4428 (Siemens/Infineon)*
- *GPM8K (Gemplus)*
- *Primeflex Store 8K (Schlumberger)*

Ces cartes dites « intelligentes », et par conséquent difficiles à mettre en œuvre sur des lecteurs « maison », offrent une capacité assez importante de 1 Ko, soit 8 192 bits.

À tout moment, chaque octet peut être protégé individuellement contre l'écriture, et cela de façon irréversible.

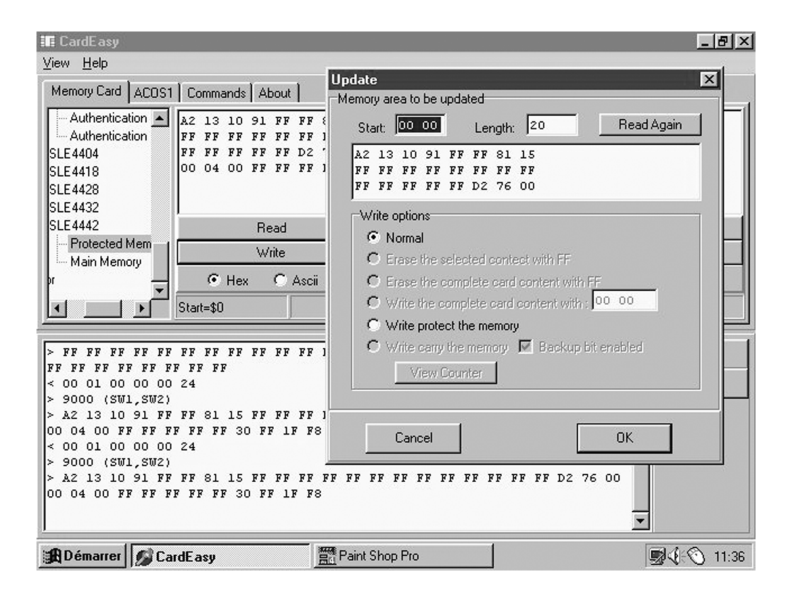
Par rapport au SLE4418, le SLE4428 dispose, en supplément, d'un système de code confidentiel (2 octets) : tant que celui-ci

n'aura pas été présenté, seules des opérations de lecture pourront être réalisées.

Bien entendu, la carte se bloque définitivement au-delà d'un certain nombre de présentations de codes erronés…

Cartes à protocole « 2 fils » (code famille 06h) :

- *SLE4432, SLE4442 (Siemens/Infineon)*
- *PCB2032, PCB2042 (Philips)*
- *GPM2K (Gemplus)*
- *Primeflex Store 2K (Schlumberger)*
- *ELEA 500 (Elea CardWare)*

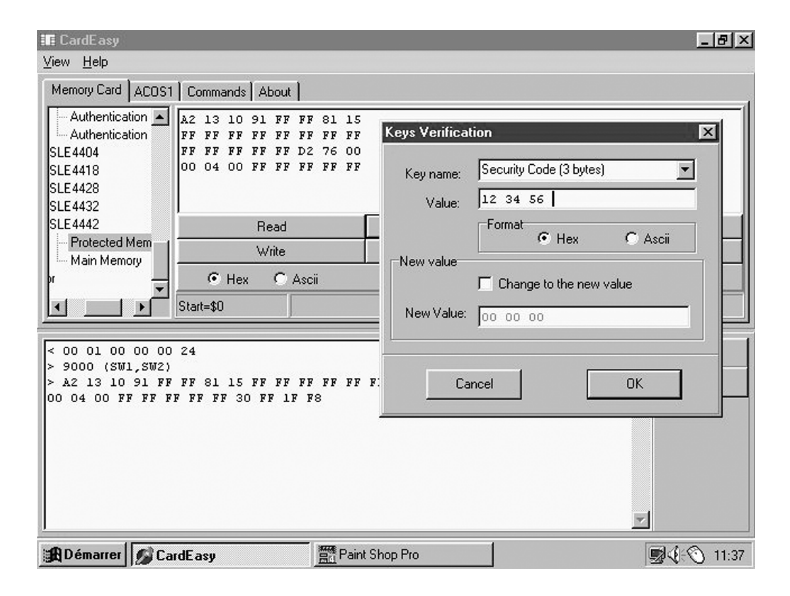


Proches des cartes à protocole « 3 fils », les cartes de cette famille utilisent un mode de communication différent (rappelant un peu l'I2C) et n'offrent qu'une capacité de 256 octets (soit 2 048 bits).

Les mécanismes sécuritaires sont également assez voisins, le code confidentiel du SLE4442 (ou PCB2042) tenant toutefois sur trois octets.

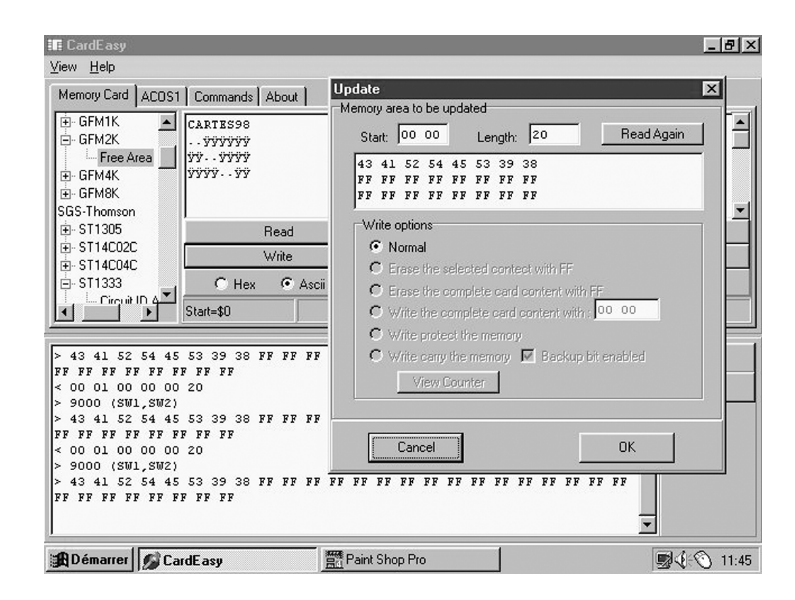
Ces cartes se prêtent bien à des applications de fidélisation, et ont notamment été adoptées par les magasins « Boots » en Grande Bretagne.

Précisons d'ailleurs que la carte ELEA 500, qui a été déployée dans de nombreuses applications extrêmement variées, n'était autre qu'une carte de cette famille, prépersonnalisée de façon particulièrement astucieuse par ELEA CardWare.



Cartes à protocole I2C (code famille 02h) :

- *AT24C01, AT24C02, AT24C04, AT24C08, AT24C16 (Atmel)*
- *ST14C02C, ST14C04C (STMicroelectronics)*
- *D2000, D4000, D8000 (Philips)*
- *GFM1K, GFM2K, GFM4K, GFM8K (Gemplus)*
- *Open 2K, Open 4K (Schlumberger)*



Les cartes de cette famille offrent un accès totalement libre aux données qu'elles contiennent, aussi bien en lecture qu'en écriture.

Organisée par octets (et non plus par bits), leur mémoire peut atteindre une capacité de 2 Ko, soit 16 384 bits.

Leur protocole de communication est conforme à la spécification I2C, la ligne SCL du bus correspondant au contact d'horloge, et la ligne SDA au contact d'entrée-sortie de données.

Le cas échéant, les cartes à puce I2C peuvent être émulées en câblant des EEPROM équivalentes, présentées en boîtier DIP 8, sur des « fausses cartes » en circuit imprimé de 8/10 mm d'épaisseur.

Moyennant la gravure d'un tel adaptateur selon le tracé de la **figure 3.1**, le lecteur « CyberMouse » peut ainsi fort bien faire office de programmateur de mémoires EEPROM série, que l'on enfichera sur un support « tulipe » câblé selon la **figure 3.2**.

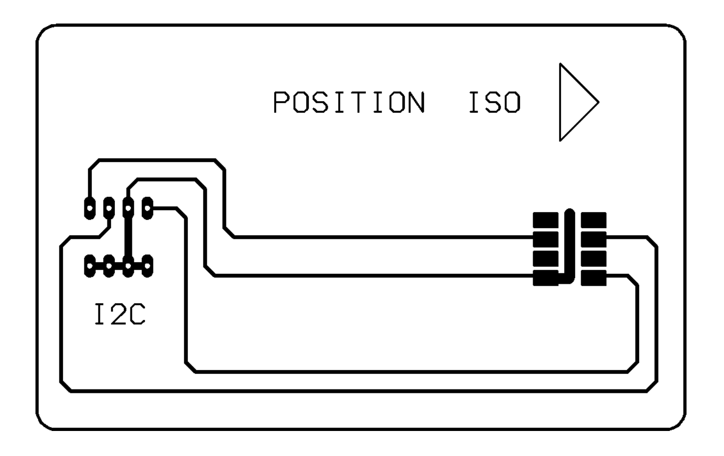


Figure 3.1. **Tracé de la « fausse carte » I2C.**

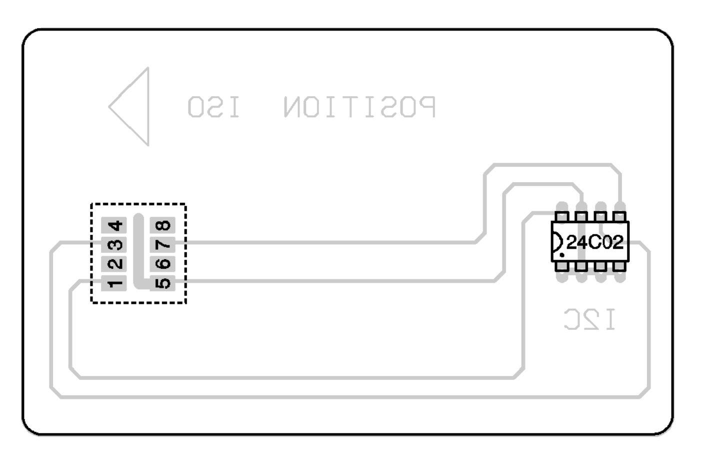


Figure 3.2. **Plan de câblage de la « fausse carte ».**

Cartes à protocole I2C étendu (code famille 07h) :

- *24C65 (Microchip, etc.)*

Au-delà de 16 kbits, le protocole I2C classique doit céder la place à une variante dite « étendue » (parfois baptisée « XIIC »).

Les cartes à puce de cette famille atteignent ainsi des capacités allant jusqu'à 8 Ko, soit 65 536 bits, comparables à celles de certaines cartes asynchrones.

Cartes à protocole Microwire (code famille 08) :

- *93CS06, 93CS46 (diverses marques)*

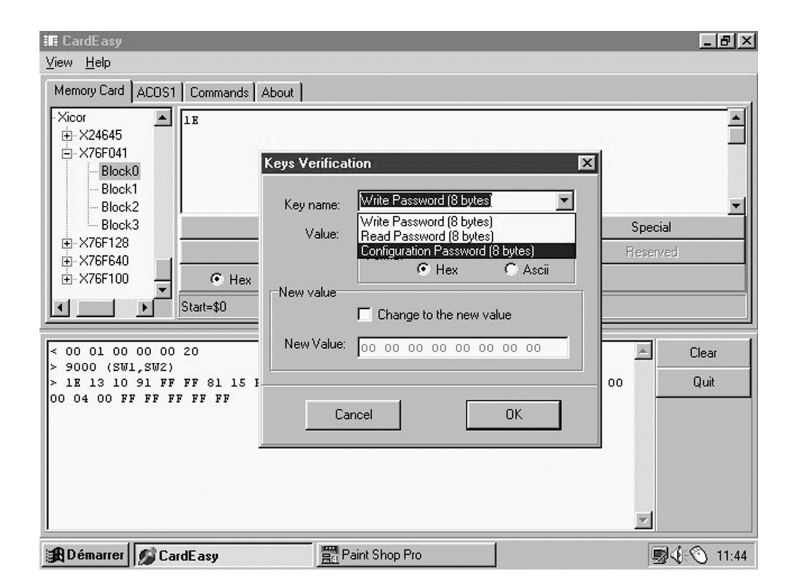
Relativement peu utilisé dans les cartes à puce, le protocole « Microwire » est un concurrent de l'I2C en matière de mémoires EEPROM série.

Les modèles supportés par le CyberMouse sont organisés en 16 registres de 16 bits, soit un total de 256 bits.

Un registre de protection permet d'interdire l'écriture dans un nombre donné d'emplacements de 16 bits, qu'il est aussi possible de protéger de façon irréversible.

Là encore, le montage de composants équivalents sur une « fausse carte » en époxy de 8/10 pourrait permettre d'émuler cette famille de cartes, ou d'utiliser le CyberMouse en tant que programmateur d'EEPROM série.

Cartes à composants Xicor X76041 ou X76F041 (code famille 0Ah), X76F128 ou X76F640 (code famille 0Eh), X76F100 (code famille 10h).



Spécialiste des EEPROM, notamment série, Xicor propose un certain nombre de mémoires sécurisées qui ne pouvaient manquer de trouver des applications dans le domaine des cartes à puce.

Présentant des caractéristiques tout à fait spécifiques, et d'ailleurs fort intéressantes, ces composants sont couverts par pas moins de trois codes famille distincts.

Bien que le fichier d'aide de CardEasy fournisse des informations relativement détaillées, on ne saurait trop conseiller de se reporter aux documentations du fabricant.

D'une façon générale, il convient d'ailleurs de noter que certaines « puces » sont produites en concurrence (parfois avec de subtiles différences) par plusieurs marques de semiconducteurs, et que les fabricants de cartes s'approvisionnent volontiers auprès de sources diversifiées.

Les télécartes de France Telecom, par exemple, sont indifféremment équipées de puces d'origine STMicroelectronics ou Texas Instruments.

En principe, CardEasy sera utilisable avec toutes les références de puces ou de cartes réunies sous un même « code famille », et avec leurs équivalents existants ou à venir, éventuellement disponibles sous d'autres marques.

Retenons par conséquent que cette liste n'est en rien limitative !

En présence d'un lecteur ACR 30, seul un sous-ensemble de cette liste serait supporté, couvrant cependant l'essentiel des cas de figure les plus intéressants (l'onglet « About » de CardEasy affiche, entre autres détails techniques, la liste des codes famille des cartes effectivement supportées par le lecteur en place).

Pour être parfaitement honnête, nous devons reconnaître que d'autres fabricants de lecteurs offrent des logiciels comparables, tel Towitoko (http://www.towitoko.de) avec son « Smart Card Editor », destiné à ses lecteurs « ChipDrive ».

Sa principale particularité est de tenter de deviner, avec des fortunes diverses, le type de carte que contient le lecteur.

3.2 UN ADAPTATEUR PC/SC POUR CARTES SYNCHRONES

Si le CyberMouse reconnaît une grande variété de cartes synchrones, bien d'autres lecteurs PC/SC supportent seulement les cartes asynchrones, I2C, et à protocole 2/3 fils.

Avec leur protocole franco-français, les télécartes T1G ou même T2G demeurent ainsi complètement à l'écart du « phénomène PC/SC », mais c'est peut-être voulu…

La vocation du présent adaptateur est donc de « déguiser » une télécarte T1G ou T2G en carte asynchrone, assurant indirectement sa compatibilité avec les lecteurs PC/SC des micro-ordinateurs fonctionnant sous Windows.

Et pour ce faire, nous allons de nouveau utiliser un microcontrôleur, mais programmé cette fois-ci en assembleur !

Chacun sait, en effet, que le PIC 16F84 se prête admirablement à la réalisation de cartes à puce asynchrones, souvent désignées sous le nom de « wafer cards ».

Quelques applications à succès dans des domaines aussi variés que la télévision à péage, la monétique, ou la téléphonie mobile sont là pour en témoigner !

Une « recette » permettant d'imiter le protocole « T = 0 » avec un PIC est d'ailleurs détaillée dans notre ouvrage *PC et cartes à puce*.

D'un autre côté, nous avons démontré au chapitre 1 qu'il était extrêmement simple de lire une télécarte avec un PIC (et l'écriture n'est guère plus compliquée !).

Toute l'originalité du présent projet consiste à réunir ces deux techniques dans un seul et même PIC, inséré entre un connecteur de cartes à puce et une « fausse carte » en circuit imprimé de 8/10 mm.

Un « prolongateur » actif, en quelque sorte, venant s'intercaler entre la télécarte et le lecteur PC/SC.

Sur le plan matériel, rien de plus simple comme en témoigne le schéma de la **figure 3.3**.

À part un condensateur de découplage du *Vcc* et une résistance de tirage sur la ligne de données de la carte synchrone, il ne s'agit que d'amener au PIC les contacts des connecteurs « mâle et femelle ».

On n'a même pas besoin d'alimentation ni de générateur d'horloge, tout cela étant fourni par le lecteur PC/SC.

À vrai dire, c'est surtout la réalisation pratique qui appelle quelques commentaires.

Deux circuits imprimés distincts devront être gravés, l'un sur de l'époxy simple face « mince » de 8/10 mm (**figure 3.4**), et l'autre sur de l'époxy simple face « ordinaire » de 16/10 mm (**figure 3.5**).

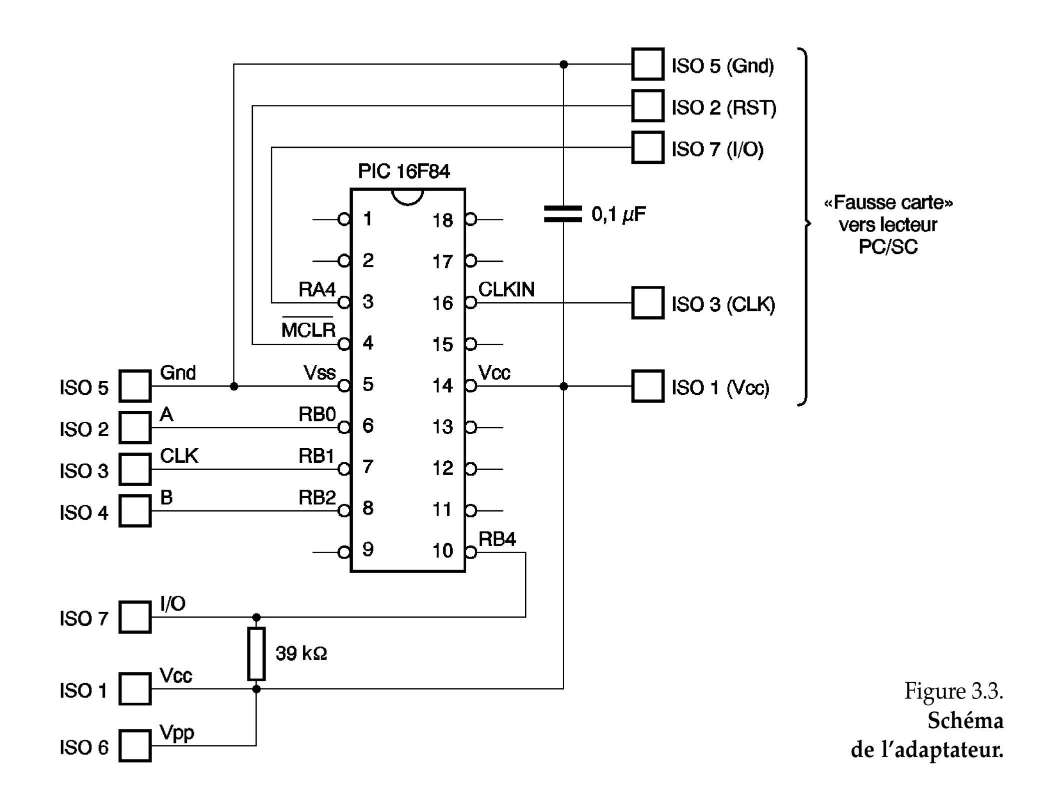


Figure 3.3. **Schéma de l'adaptateur.**

Selon la profondeur d'insertion de la carte dans le lecteur utilisé, la longueur de celle-ci pourra d'ailleurs éventuellement être réduite, mais ce n'est nullement une obligation.

Le câblage de la « fausse carte » se limite à la mise en place d'un support DIL 18 broches « à wrapper ».

Après soudage, les longues queues très rigides de celui-ci seront insérées une nouvelle fois dans les trous de la seconde carte, préalablement équipée de ses composants selon le plan de la **figure 3.6**.

On notera que le contact « présence carte » du connecteur de carte à puce n'étant pas mis à contribution, il est possible d'utiliser aussi bien un modèle à contact normalement ouvert (N.O.) que normalement fermé (N.F.), au hasard des prix promotionnels pratiqués ici ou là.

Nous avons tenu compte de cette opportunité dans le pastillage du circuit imprimé, qui peut accepter des modèles de différentes marques (ITT-CANNON, ALCATEL, etc.)

Seuls les huit balais de contact correspondant aux cartes avec puce en position « ISO » sont connectés, ce qui est compatible avec toutes les T2G et avec les dernières séries de T1G.

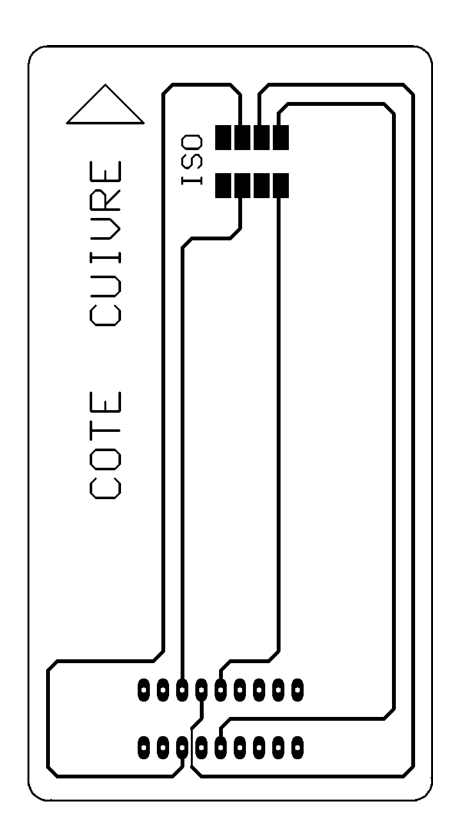


Figure 3.4.
Tracé du circuit imprimé 8/10 mm.

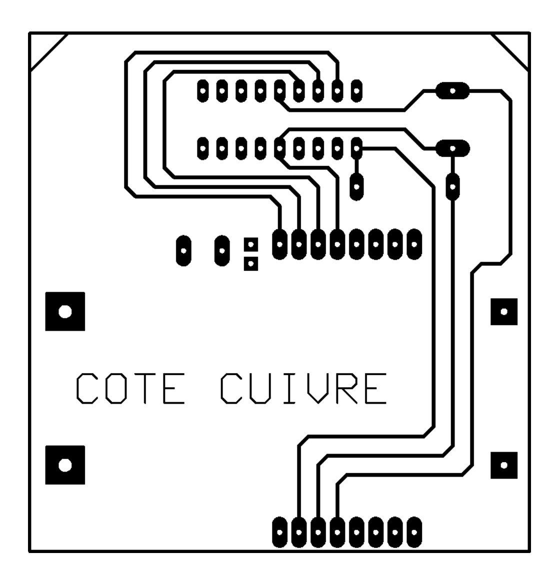


Figure 3.5.
Tracé du circuit imprimé 16/10 mm.

Dans le cas, peu probable, où l'on tiendrait absolument à pouvoir opérer aussi sur de « vieilles » cartes avec puce en position

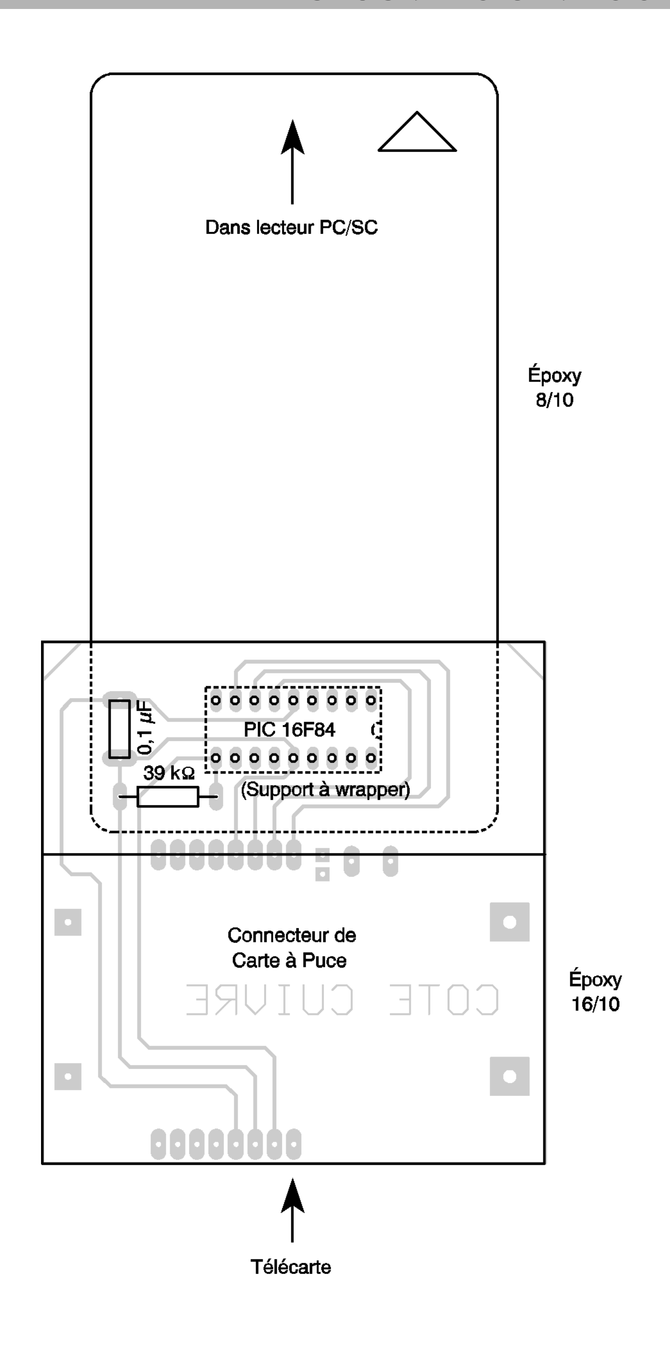
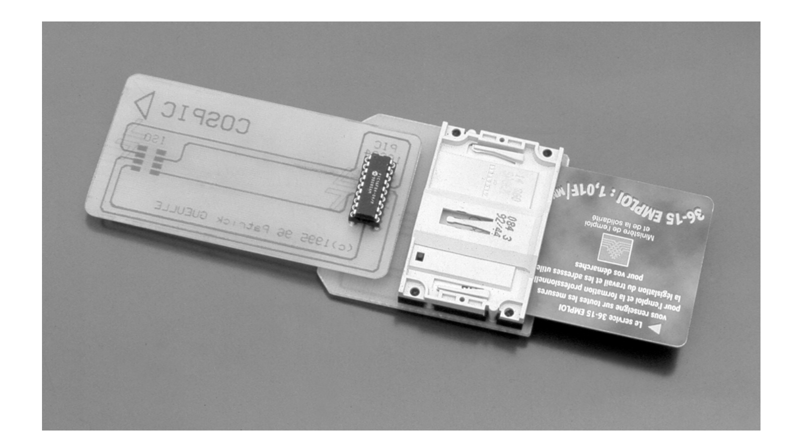


Figure 3.6.
Plan de câblage.

Liste des composants
PIC 16F84
à programmer
(SYNC.HEX)
1 × 39 kΩ
1 × 0,1 µF
connecteur
de cartes à puce
support à wrapper
DIL 18

AFNOR, il suffirait de souder huit courts fils de câblage côté cuivre, pour mettre en parallèle les deux jeux de balais que l'on trouve encore sur certains connecteurs (les pastilles correspondantes sont prévues). Tel est le prix à payer pour échapper à un circuit double face…



Chacun aura compris que tout le « secret » de ce montage réside dans le logiciel (fichier SYNC.HEX) qui devra être programmé dans le PIC.

Le code source de celui-ci associe la partie spécifique à la lecture-écriture des télécartes (SYNC.ASM), au jeu de routines « T = 0 » décrit dans notre ouvrage *PC et cartes à puce*.

Du côté « asynchrone », le montage se comporte comme une carte qui ne reconnaîtrait que les deux commandes ISO suivantes :

- `22 44 00 00 00` (avance d'un bit avec lecture) ;
- `22 22 00 00 00` (mise à 1 du bit courant).

Cela correspond respectivement aux « micro-instructions » UP et PGM de la **figure 3.7**, la micro-instruction RESET étant, pour sa part, automatiquement exécutée toutes les fois que le lecteur PC/SC effectue lui-même un reset.

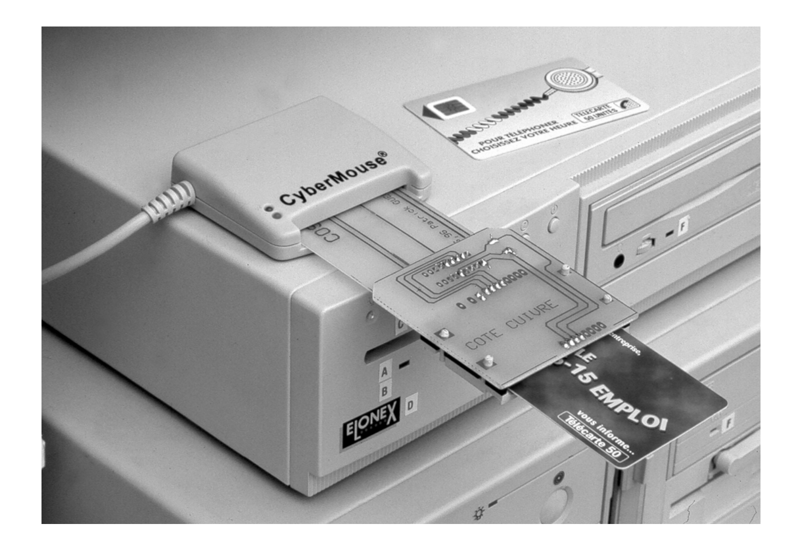
ISO 2 (A)	ISO 4 (B)	ISO 3 (CLK)	Micro - Instruction
0	0		Reset
0	1		(44) UP
1	1		(22) PGM (0 → 1)

Figure 3.7.
Les micro-instructions mises en œuvre.

Précisons que la durée des impulsions d'horloge (CLK) appliquées à la télécarte (de quelques microsecondes à quelques millisecondes selon l'opération à effectuer) est calibrée par le PIC, à partir de la fréquence d'horloge émise par le lecteur PC/SC (de l'ordre de 3,58 MHz).

Il en résulte que le « timing » est systématiquement correct, quelle que soit la fréquence d'horloge du PC utilisé, contrairement au cas des programmes GWBasic sur port parallèle, qui pouvaient nécessiter un étalonnage.

Étudions donc maintenant ce qui se passe lorsque le montage, préalablement muni d'une télécarte, est inséré dans un lecteur PC/SC.



Dès l'introduction de la « fausse carte », le lecteur PC/SC commande un « reset » général, ce qui positionne la télécarte sur son premier bit.

Simultanément, le PIC émet une réponse au reset minimale (`3F 00`) qui indique au lecteur PC/SC quel protocole utiliser pour la suite des opérations (T = 0, convention inverse, 9 600 bps).

Dès lors, tout se passera par enchaînement de commandes ISO 7816, et par interprétation des seuls comptes rendus SW1SW2 retournés après leur exécution (pour gagner du temps, nous n'avons pas utilisé de champ « données » dans ces commandes).

La commande `22 44 00 00 00` doit retourner SW1 = 90h, SW2 contenant l'état du bit lu, avant mais aussi après son exécution.

`90A5` indiquerait ainsi que le bit lu avant l'exécution de la micro-instruction UP était à 1, et que celui lu après est à 0.

`9055` signifierait que le bit lu était à 0 et y est resté, `90AA` qu'il était à 1 et y est resté, tandis que `905A` rendrait compte d'un passage de 0 à 1.

La commande `22 22 00 00 00`, par contre, doit toujours renvoyer SW1SW2 = `9022`, que l'opération d'écriture ait réussi ou non (la vérification se fera généralement à l'occasion de la prochaine commande `22 44 00 00 00`).

Une micro-instruction PGM isolée ne peut, en effet, que mettre à 1 un bit qui était à 0, et jamais le contraire.

Tel qu'il est proposé, ce montage ne permet d'ailleurs l'écriture que dans les T2G, pour la bonne et simple raison que les lecteurs PC/SC ne fournissent pas le *Vpp* de 21 V nécessaire à l'écriture dans les T1G (pour ce genre de besogne, un lecteur « maison » ferait beaucoup mieux l'affaire !).

De toute façon, l'opération est bien plus intéressante sur les T2G, en raison de cette fameuse variante dite « avec retenue ».

Ne faisant pas l'objet d'une commande spécifique, cette micro-instruction est réalisée en enchaînant deux commandes « PGM » sur un même bit.

Exécutée dans certaines zones du « plan mémoire » de la T2G, cette opération permet d'effacer (remettre à 0) huit bits d'un coup, au prix de la mise à 1 d'un seul bit dans l'octet de poids supérieur.

C'est ce mécanisme qui sert à faire tourner le compteur de type « boulier », capable de comptabiliser plus de trente mille unités avec seulement 40 bits.

Naturellement, le maximum semble avoir été fait pour interdire tout ce qui pourrait s'apparenter, de près ou de loin, à un « rechargement » !

À partir du moment où une télécarte introduite dans cet adaptateur est vue comme une carte asynchrone, sa compatibilité se trouve assurée avec le kit de développement logiciel de la BasicCard (voir chapitre 2).

C'est donc en ZCBasic que nous avons écrit les deux logiciels applicatifs de ce projet.

SYNC.BAS se borne à lire 512 bits consécutifs dans toute télécarte, c'est-à-dire la totalité de l'espace adressable d'une T2G.

Dans le cas d'une T1G, on lirait par contre deux fois de suite les 256 bits qu'elle contient.

En plus d'être affiché à l'écran, le résultat de la lecture s'enregistre dans un fichier CARTE.CAR, entièrement compatible avec les programmes de nos ouvrages *Cartes à puce – Initiation et applications* et *PC et cartes à puce*.

Ceux-ci permettraient donc, le cas échéant, de « décrypter » les données lues dans une T1G, une T2G, voire une télécarte étrangère de type « Eurochip », par exemple pour déterminer s'il y reste ou non quelque crédit d'unités.

Bien que ce logiciel soit destiné à fonctionner, en priorité, sur un lecteur PC/SC, nous n'avons pas prévu l'habituelle ligne « `ComPort=101` ».

Cela nous procure l'occasion de dévoiler une autre façon de déclarer le lecteur de cartes à puce à utiliser.

Avant de lancer SYNC.EXE, on tapera (dans une fenêtre MS-DOS) la ligne de commande `SET ZCPORT = 101` si le lecteur à utiliser fonctionne en mode PC/SC, `SET ZCPORT = 2` si le lecteur « Cyber-Mouse » du kit BasicCard est branché sur COM2:, et ainsi de suite.

Cette façon de procéder permet de ne compiler qu'un seul et unique exécutable, quel que soit le lecteur utilisé, d'autant que la commande `SET ZCPORT` peut fort bien être placée dans l'autoexec.bat du PC.

Notre seconde application (MANIP.BAS) est plus ambitieuse, dans la mesure où elle permet d'opérer n'importe quelle manœuvre sur n'importe quel bit d'une T2G, à l'exception du tout premier (qui est néanmoins lu, et de toute façon inaltérable).

C'est, en fait, la version PC/SC de l'utilitaire « MANIPT2G. BAS » de notre livre *Cartes à puce – Initiation et applications*.

Son ergonomie est quasiment identique, seul son fonctionnement étant un peu plus lent puisqu'une commande ISO 7816 entière (soit 70 bits au lieu d'un seul) doit être acheminée à 9 600 bps lors de chaque lecture ou écriture d'un bit de la télécarte.

Chaque appui sur la barre d'espace du clavier fait avancer, rappelons-le, la position de lecture d'un bit, la touche + permettant de le mettre à 1 s'il était encore à 0.

La touche –, pour sa part, commande une mise à 1 « avec retenue », qui n'aura une action spécifique que dans les zones prévues à cet effet (mais rien n'interdit d'essayer ailleurs, « juste pour voir »).

D'une façon générale, après une écriture avec retenue, il vaudra mieux quitter le programme (touche ESCape) puis le relancer pour juger de l'effet produit.

Les spécifications techniques d'un composant très proche de celui équipant les T2G précisent, en effet, qu'une lecture de bit,

effectuée juste après une écriture avec retenue, ne donne pas de résultat significatif, et cela semble bien se vérifier.

À tous points de vue, il vaut donc mieux opérer un reset et reprendre la lecture au début.

On le voit, l'association de cet adaptateur et d'un lecteur PC/SC fournit, tout spécialement aux heureux possesseurs du kit Basic-Card, la possibilité de développer toutes sortes d'applications autour des télécartes (épuisées ou non), en toute indépendance vis-à-vis des caractéristiques du PC utilisé, ou de son système d'exploitation.

Bien entendu, cette démarche est transposable à tout autre lecteur de cartes à puce asynchrones, et à n'importe quel autre environnement de développement, sans changer quoi que ce soit aux parties matérielle ou logicielle de l'adaptateur.

Moyennant cette fois quelques transformations, la même approche pourrait aussi convenir à d'autres familles de cartes synchrones, dont on aurait préalablement étudié le jeu de « micro-instructions ».

Et précisément, nous y arrivons !

3.3 UN « ESPION » DE CARTES À PUCE SYNCHRONES

Nous avons vu qu'intercepter le dialogue entre une carte à puce asynchrone et son terminal ne présente pas de grandes difficultés.

Paradoxalement, il est bien plus délicat de faire de même avec une carte synchrone, pourtant beaucoup moins perfectionnée.

En protocole asynchrone (T = 0 ou T = 1), tout le dialogue entre la carte à puce et le terminal dans lequel elle est insérée se fait sur un seul fil (le contact ISO 7 du micromodule), à un rythme qui dépend étroitement de la fréquence du signal d'horloge appliqué sur le contact ISO 3 (CLK).

Une certaine similitude avec le protocole RS232 permet d'obtenir des résultats fort convaincants en « détournant » astucieusement un port série de PC (voir chapitre 2).

Dans le cas des cartes à puce synchrones (télécartes, I2C, 2/3 fils, etc.), on sait qu'il existe de nombreux protocoles différents, qui mettent en œuvre des séquences d'états plus ou moins compliquées sur plusieurs contacts à la fois.

Le principal point commun est que toutes les opérations significatives se font à l'occasion d'une transition, soit positive soit néga-

tive, du signal d'horloge, autrement dit en strict « synchronisme » avec celui-ci.

Chaque famille de cartes « synchrones » reconnaît ainsi un jeu de « micro-instructions » qui lui est propre, pour lire ou écrire un bit, atteindre une adresse donnée, présenter un code confidentiel bit par bit, etc.

En principe, il faudrait au minimum un analyseur logique pour inspecter finement le dialogue entre une telle carte et son lecteur, ce qui peut s'avérer fort instructif.

Comme la fréquence d'horloge mise en jeu est presque toujours inférieure à 50 kHz (c'est une exigence de la norme ISO 7816-3), il est cependant possible d'imaginer des solutions plus simples, voire originales, et pourquoi pas élégantes.

Dès que le dialogue avec une carte à puce atteint un certain degré de complexité (et cela va vite !), il devient difficile de procéder à une interprétation en temps réel.

Les milliers d'octets échangés, par exemple, entre un téléphone GSM et sa carte SIM, doivent nécessairement être enregistrés, puis analysés à tête reposée (voir notre ouvrage *Téléphones portables et PC*).

On peut songer à pratiquer ce genre d'enregistrement au moyen d'un PC, mais également d'un dispositif autonome et peu encombrant, dont on « videra » ultérieurement la mémoire dans le PC utilisé pour son « dépouillement ».

Dans le cas particulier des cartes à puce synchrones, il est nécessaire de disposer d'un outil capable d'enregistrer plusieurs « voies » en parallèle, puisqu'en plus de l'horloge, de l'alimentation, et de la masse, certaines cartes peuvent gérer jusqu'à quatre contacts du micromodule (et notamment les « RFU » ISO 4 et ISO 8).

Le schéma de la **figure 3.8** résout ce problème au moyen d'une mémoire « Zeropower » (48Z08), capable de stocker 8 192 mots de 8 bits : ce sera amplement suffisant !

Elle est associée à un classique compteur 4040, mais de façon plutôt inhabituelle, puisque celui-ci ne pilote que ses lignes d'adresse A1 à A12.

La ligne d'adresse de poids faible (A0), pour sa part, est directement attaquée par le signal d'horloge de la carte.

Il en résulte que les états des contacts ISO 2, 4, 7, et 8 seront mémorisés dans les deux états possibles du contact d'horloge

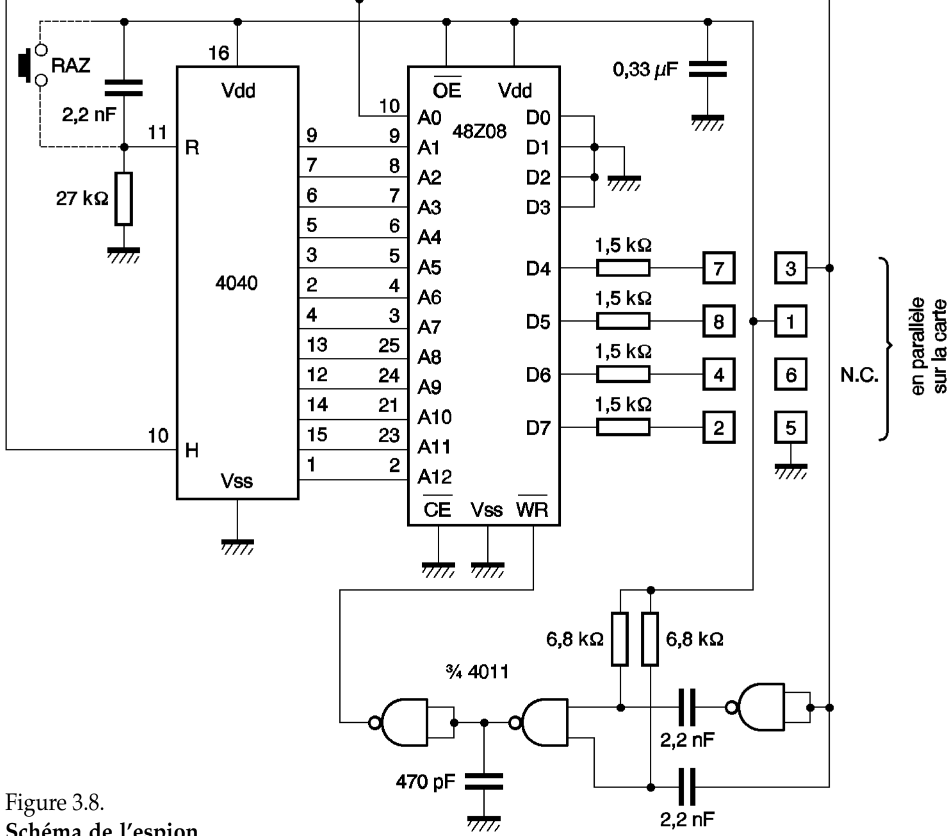


Figure 3.8.
Schéma de l'espion de cartes synchrones.

(ISO 3) : dans les adresses paires quand ISO 3 est à 0, et dans les adresses impaires quand il est à 1.

C'est absolument indispensable, car avec une même carte, il peut se passer des choses très différentes lors d'un front avant ou arrière du signal d'horloge !

Le compteur d'adresses étant attaqué par ce même signal d'horloge, un artifice technique est naturellement nécessaire pour déclencher une écriture en mémoire (impulsion négative sur $\overline{WR}$) un peu après chaque front avant et arrière.

Ce résultat est obtenu en différentiant à la fois le signal d'horloge et son complément logique, puis en appliquant une fonction ET à ces deux canaux.

Les valeurs des réseaux RC différentiateurs sont fixées par la limite supérieure de la fréquence d'horloge (50 kHz), et il en va de même pour le condensateur (470 pF) chargé d'augmenter artificiellement le « temps de propagation » du circuit.

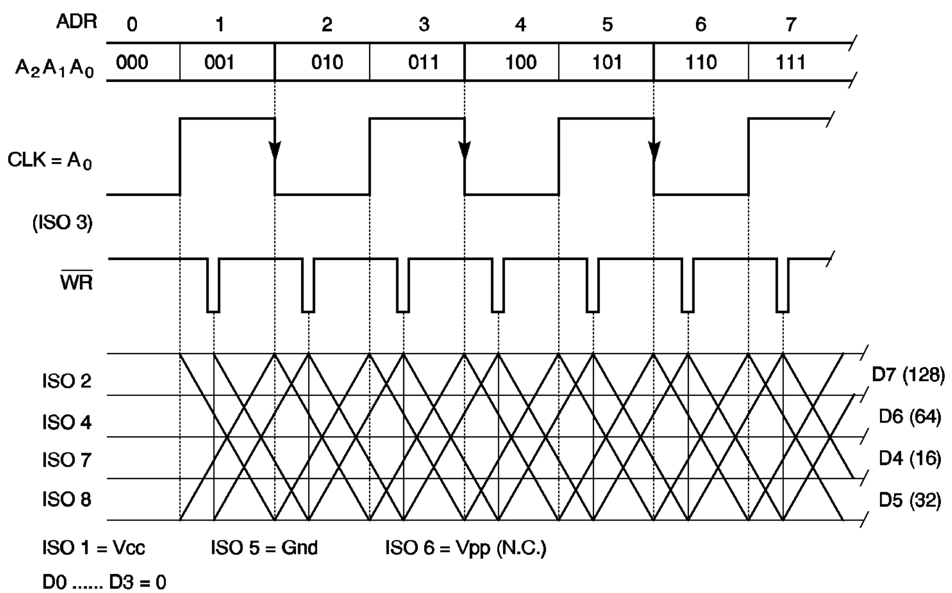


Figure 3.9.
Le chronogramme détaillé.

Le chronogramme de la **figure 3.9** montre comment on arrive ainsi à commander la mémorisation à un instant où les niveaux sont stables, condition nécessaire à l'acquisition de données présentant toutes garanties de validité.

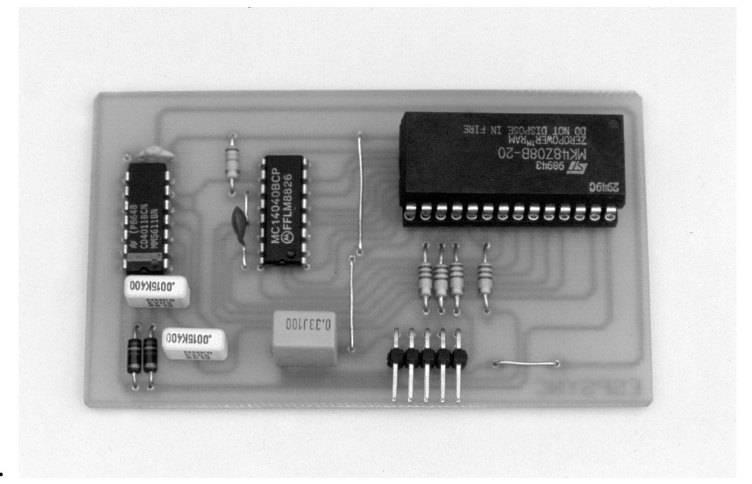
Une exception qui confirme la règle est le premier octet de la 48Z08, dont la valeur ne sera généralement pas significative.

En effet, la remise à zéro du compteur d'adresses ne se fait pas à partir du contact « Reset » de la carte à puce (dont on a besoin d'enregistrer l'activité, souvent intéressante), mais lors de la mise sous tension du montage (en même temps que la carte, et donc avant le « reset » de cette dernière).

Selon les cas, on peut donc commencer l'enregistrement à la seconde adresse de la mémoire et non pas à la première, qui contiendra alors une valeur plus ou moins aléatoire.

Il suffit de le savoir, et d'en tenir compte lors de l'interprétation de l'enregistrement.

Pour les cas difficiles, il est d'ailleurs possible d'ajouter un poussoir de remise à zéro (RAZ) manuelle. Il permettrait, par exemple, de « court-circuiter » un processus d'initialisation que l'on aurait déjà étudié en long et en large, pour n'enregistrer que ce qui se passe lors d'une opération bien particulière, sciemment déclenchée au moment opportun.



Le circuit imprimé principal de l'espion.

L'intégralité du montage tient sur un circuit imprimé simple face, dont la **figure 3.10** fournit le tracé, assez dense mais ne présentant pas de véritable difficulté de gravure.

Tout au plus fera-t-on attention aux trois fines pistes qui passent entre les pastilles du 4040, mais il s'agit là d'une technique courante.

La mise en place des composants (dont trois straps) se fera selon le plan de la **figure 3.11**, en prenant la précaution de choisir un support de bonne qualité pour la mémoire 48Z08, que l'on aura souvent l'occasion de débrocher et rembrocher.

Sans aller jusqu'à conseiller un support à force d'insertion nulle (ZIF), nous recommanderons un modèle à contacts « tulipe », autant que possible dorés.

Afin d'éviter tout risque d'endommagement, à la longue, des broches de la 48Z08, on pourra (si ce n'était déjà fait !) installer celle-ci à demeure dans un second support « tulipe » neuf (et donc un peu « dur »).

Ce sont alors les broches de celui-ci, nettement plus rigides, que l'on introduira tour à tour dans le support du montage, et dans celui du programmateur d'EPROM dont nous allons bientôt découvrir le rôle.

En cas d'accident, on aura juste à remplacer le support intermédiaire, les broches de la mémoire restant absolument intactes.

On s'étonnera peut-être de l'absence de tout branchement d'alimentation sur ce montage. En fait, il « vole » le 5 volts dont il a

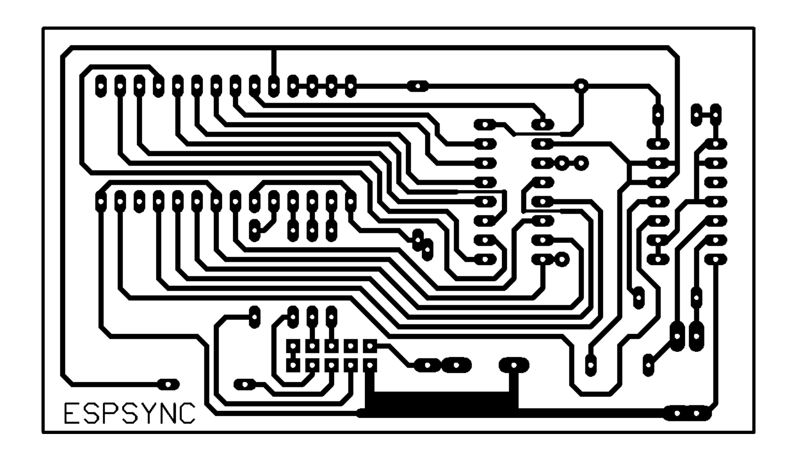


Figure 3.10.
Tracé du circuit imprimé.

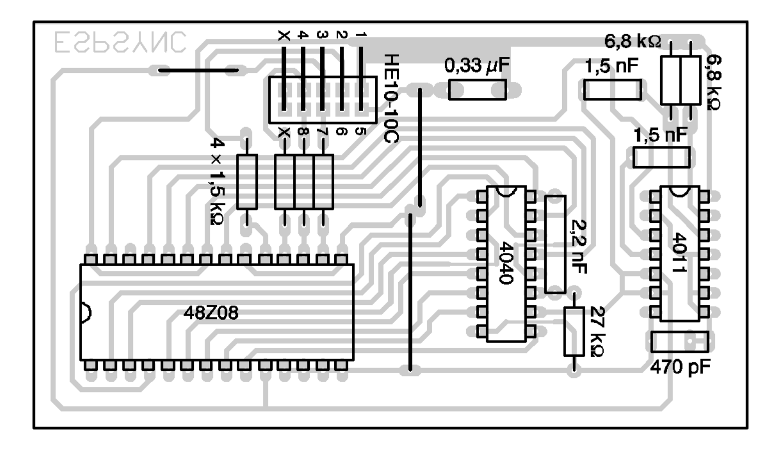


Figure 3.11.
Plan de câblage.

Liste des composants
1 × MK48Z08
ou équivalent
(Radiospares)
1 × 4040
1 × 4011
4 × 1,5 kΩ
2 × 6,8 kΩ
1 × 27 kΩ
2 × 15 kΩ (sur la « fausse carte »)
1 × 0,33 µF
2 × 1,5 nF
1 × 470 pF
1 × 2,2 nF
support « tulipe » 28 broches
barrette sécable à doubles picots carrés coudés
(ou 2 embases HE10 à 10 contacts sans verrous)

besoin sur le contact Vcc (ISO 1) de la carte, ce qui assure du même coup le fonctionnement de son circuit de RAZ automatique.

La mise en parallèle de l'espion sur la liaison carte-terminal nécessite l'usage d'une connectique très classique, décrite au chapitre 2.

Celle-ci se compose d'un connecteur de cartes à puce (desservant impérativement l'intégralité des 8 contacts du micromodule, dont on vérifiera la compatibilité du positionnement ISO ou AFNOR), d'une « fausse carte » en circuit imprimé de 8/10, et d'un câble plat muni de trois fiches HE10 à dix contacts, toutes serties dans le même sens et donc interconnectées « fil à fil ».

Naturellement, chacun pourra être amené à fabriquer sa propre variante de « fausse carte », selon les exigences dimensionnelles des lecteurs dans lesquels il sera prévu de l'introduire.

La **figure 3.12** suggère ainsi un tracé allongé, pouvant être introduit dans les lecteurs qui « avalent » la totalité de la carte, et

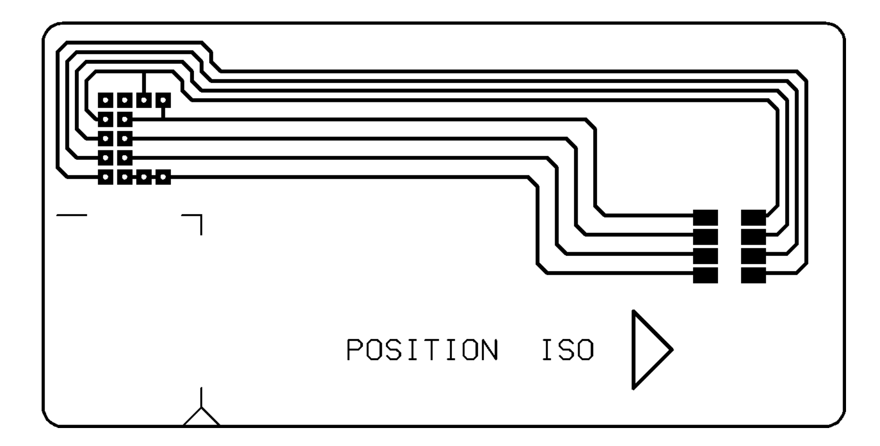
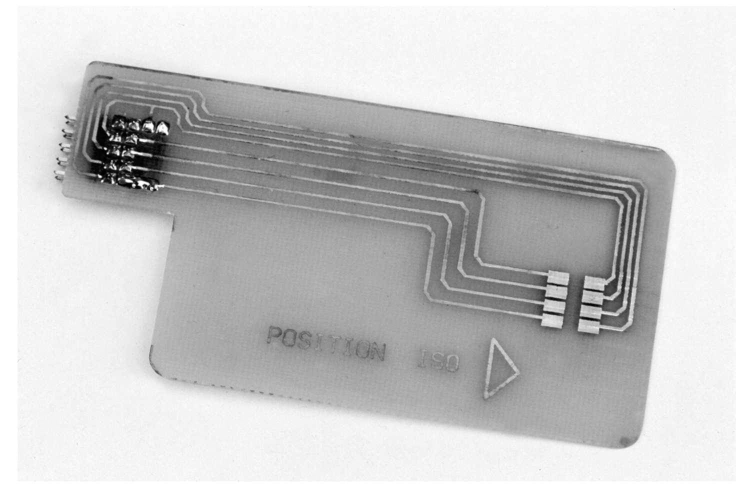


Figure 3.12.
Tracé de la « fausse carte ».



Le côté cuivre de la « fausse carte ».

comporte même une zone pouvant être évidée pour libérer au maximum son pourtour.

Le plan de câblage de la **figure 3.13** prévoit, par ailleurs, l'implantation de deux résistances de 15 000 Ω, dont le but est d'éviter de laisser « en l'air » les contacts ISO 8 et ISO 4, inutilisés sur certaines cartes.

Normalement, le lecteur est censé les mettre à la masse, mais « sur le terrain », on rencontre un peu tout et n'importe quoi…

Précisons tout de même que cette « fausse carte » n'a, en aucun cas, vocation à être introduite dans des automates publics, tels que ces parcmètres qui présentent systématiquement le code de rechargement de la carte, même lors d'une simple opération de lecture d'un crédit… épuisé.

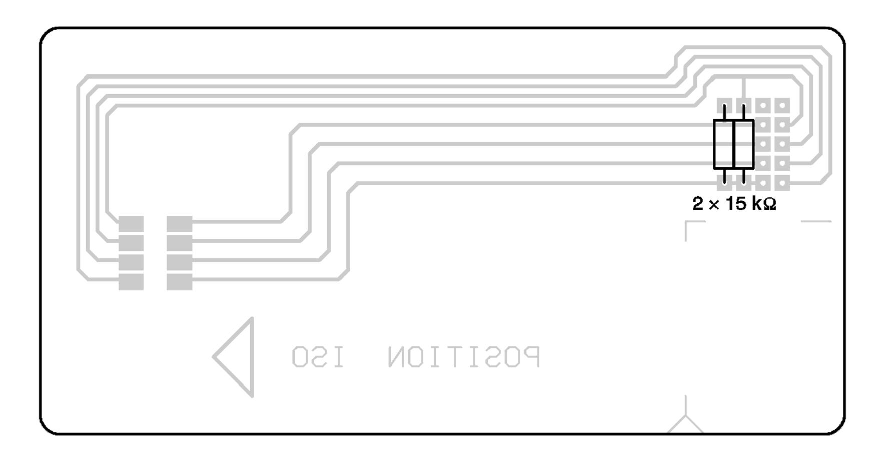
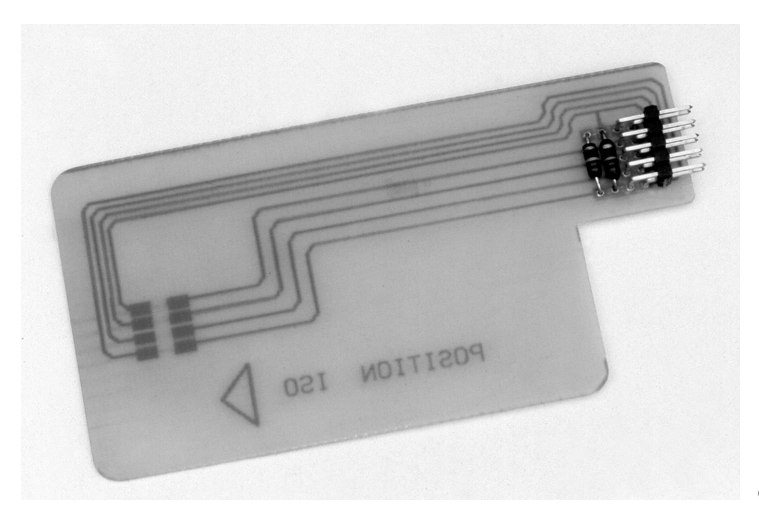


Figure 3.13.
Plan de câblage.



Le côté composants de la « fausse carte ».

On se doute, en effet, qu'ils doivent être équipés de « détecteurs de métaux » leur permettant de rejeter les cartes munies de telles connexions vers l'extérieur, et donc potentiellement suspectes !

Cela étant posé, il reste maintenant à exploiter les enregistrements ainsi réalisés en tout bien tout honneur.

Typiquement, la mémoire 48Z08 a toujours été un outil de développeur d'applications à base d'EPROMs (voir notre ouvrage *Composants électroniques programmables sur PC*), et il y a donc gros à parier que si l'on en possède une, on dispose également d'un programmateur *ad hoc*.

Il y avait là une excellente occasion de simplifier notre montage, en n'y incorporant aucune fonction de relecture de la mémoire.

Cette opération se fera tout simplement en transportant celle-ci (puisqu'elle est « non volatile » !) sur un programmateur capable de lire les 27C64, dont le brochage est 100 % compatible en mode « lecture ».

Rappelons que notre ouvrage précité contient les plans et logiciels d'un programmateur extrêmement simple, convenant parfaitement à cet usage.

À ce stade, il est intéressant de commenter quelques échantillons d'enregistrements, réalisés sur un certain nombre de cartes avec lesquelles le logiciel « CardEasy » nous a déjà donné l'occasion de faire plus ample connaissance.

Ils ont été effectués soit dans un lecteur « maison » (voir *Cartes à puce – Initiation et applications*), soit dans un lecteur « CyberMouse » placé, précisément, sous le contrôle du logiciel « CardEasy ».

Précisons que les octets lus dans la Zeropower sont présentés en format décimal, le poids 16 correspondant à la ligne d'entrée-sortie (contact ISO 7). Un niveau haut sur cette ligne sera ainsi matérialisé par une valeur 16 si tous les autres contacts sont au niveau bas, ou encore par 80 si le contact ISO 4 (poids 64), et lui seul, est simultanément au niveau haut.

Cela correspond à deux variantes courantes du processus de remise à zéro de la carte.

On ne manquera pas de remarquer, également, la différence de comportement entre les télécartes (T1G) d'origine Texas Instruments (TMS 3561) et STMicroelectronics (ST1200) : les premières présentent le bit lu, sur le contact ISO 7, dès que le contact d'horloge (ISO 3) arrive au niveau haut.

Les secondes attendent, par contre, le retour de celui-ci au niveau bas.

Là se situe l'explication de ces très nettes « colonnes » de valeurs 80, qui correspondent à cet état transitoire 1 sur ISO 7 pendant qu'ISO 3 est à 1, et cela, quel que soit l'état du bit adressé.

Et ne parlons même pas de ce qui se passe avec les lecteurs (et ils sont légion !) qui ne délivrent pas de 5 volts au contact Vpp (ISO 6)…

Bien que plus complexe, le protocole des cartes « I2C » est également « décortiqué » dans ses moindres détails par notre montage : on peut suivre très nettement les états de la ligne SDA (poids 16) au fil des transitions, aussi bien positives que négatives, de la ligne SCL (horloge).

TMS 3561 (lecteur « maison ») :
Décodage des 32 premiers bits

```
 0  16  16  16  16  80  80  64         1 1
64  64  64  80  80  80  80  80     0 0 1 1
80  80  80  64  64  64  64  64     1 1 0 0
64  64  64  64  64  80  80  64     0 0 0 1
64  64  64  64  64  64  64  64     0 0 0 0
64  64  64  64  64  64  64  80     0 0 0 0
80  80  80  64  64  64  64  64     1 1 0 0
64  64  64  64  64  80  80  80     0 0 0 1
80  64  64  80  80  64  64  80     1 0
80  80  80  64  64  80  80  80
80  80  80  80  80  64  64  64
```

ST1200 (lecteur « maison ») :
Décodage des 32 premiers bits

```
 0 208  16  16  16  80  64  80         1 0
64  80  64  80  80  80  64  80     0 0 1 0
64  80  64  80  80  80  64  80     0 0 1 0
64  80  64  80  64  80  64  80     0 0 0 0
64  80  64  80  64  80  64  80     0 0 0 0
80  80  64  80  64  80  64  80     1 0 0 0
64  80  64  80  64  80  64  80     0 0 0 0
64  80  64  80  64  80  64  80     0 0 0 0
80  80  64  80  80  80  64  80     1 0
```

SLE 443x (CyberMouse) :
Lecture des 32 premiers bits

```
16 128 128  16   0   0  16  16       1 0 1
 0   0  16  16   0   0   0   0     0 1 0 0
 0   0   0   0   0   0  16  16     0 0 0 1
 0   0   0   0   0   0  16  16     0 0 0 1
 0   0  16  16  16  16  16  16     0 1 1 1
16  16   0   0  16  16  16  16     1 0 1 1
16  16  16  16   0   0   0   0     1 1 0 0
16  16   0   0  16  16   0   0     1 0 1 0
 0   0   0 128 128  16   0   0     0
```

SLE 4404 (CyberMouse) :
Lecture des 16 premiers bits (810Ch)

```
64 208  80  64  64  64  64  64       1 0 0  (1h = 0001)
64  64  64  64  64  64  64  64     0 0 0 0  (8h = 1000)
80  80  64  64  64  64  80  80     1 0 0 1  (Ch = 1100)
80  80  64  64  64  64  64  64     1 0 0 0  (0h = 0000)
64  64  64  64  80  80  64  64     0
```

ST1301 (CyberMouse) :
Présentation code PIN (ABCDh) sans Vpp

80	80	80	80	80	80	80	80	
80	80	80	80	80	80	80	80	
80	80	80	80	80	80	80	64	
64	64	64	64	80	80	80	80	0 0 1 1 (Ah = 1100)
80	80	64	64	80	80	80	64	1 0 1 1 (Bh = 1101)
64	64	80	80	64	64	80	80	0 1 0 1 (Ch = 1010)
80	80	80	80	64	64	80	80	1 1 0 1 (Dh = 1011)
80	96	64	112	80	208	80	80	

SLE 4442 (CyberMouse) :
Lecture des 16 premiers bits (A213 h)

0	144	144	0	16	16	0	0	0 1 0 (2h = 0010)
0	0	0	0	16	16	0	0	0 0 1 0 (Ah = 1010)
16	16	16	16	16	16	0	0	1 1 1 0 (3h = 0011)
0	0	16	16	0	0	0	0	0 1 0 0 (1h = 0001)
0	0	0	0	0	0	0	0	0

Carte I2C 2K (D2000, GFM2K) :
Lecture séquentielle des 24 premiers bits (434152h)

16	16	0	16	16	0	0	16	START
16	0	0	0	0	0	0	0	Select (W) 1010 0000
0	0	0	0	16	0	0	0	
0	0	0	0	0	0	0	0	Address 0000 0000
0	0	0	0	0	0	16	16	START
0	16	16	0	0	16	16	0	Select (R) 1010 0001
0	0	0	0	0	0	0	16	
0	0	0	0	16	16	0	0	. 0 1 0 (4h = 0100)
0	0	0	0	0	0	16	16	0 0 0 1 (3h = 0011)
16	16	16	0	0	0	16	16	1 . 0 1 (4h = 0100)
0	0	0	0	0	0	0	0	0 0 0 0 (1h = 0001)
0	0	16	16	16	0	0	0	0 1 . 0
16	16	0	0	16	16	0	0	1 0 1 0 (5h = 0101)
0	0	16	16	0	0	16	0	0 1 0 . (2h = 0010)

S'agissant d'un protocole « 2 fils », aucun autre contact n'entre en jeu, et on n'enregistre très logiquement que des valeurs 16 ou 0, un peu plus délicates à interpréter.

Il y a donc vraiment là une inépuisable source de découvertes, parfois inespérées !

Un simple exemple : le secret est jalousement gardé, par les fabricants, sur le fonctionnement du mécanisme d'authentification des cartes Eurochip (les télécartes européennes, et notamment allemandes).

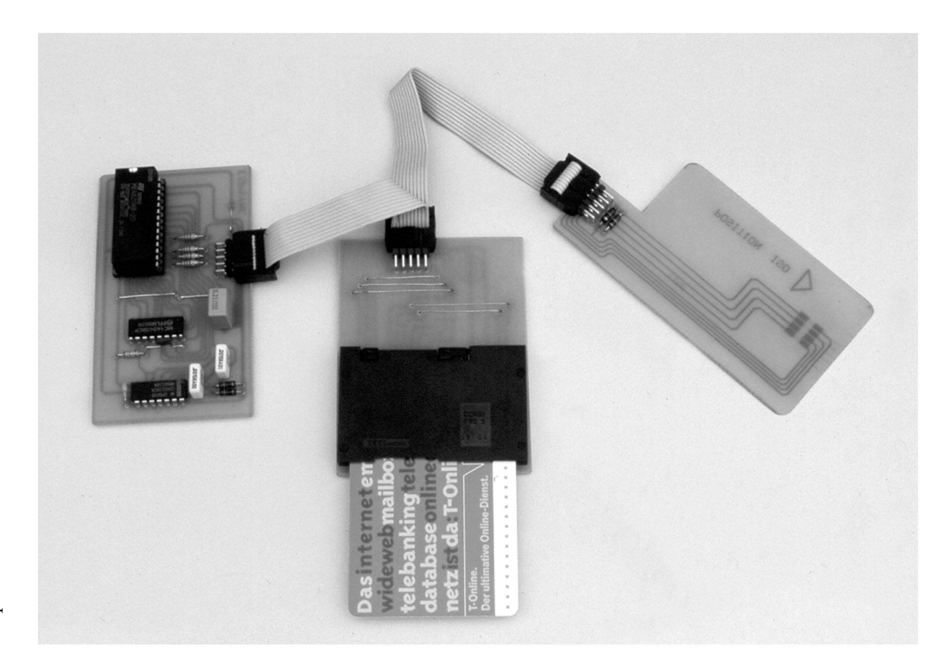
Il est pourtant mis en œuvre dans le CyberMouse, ce qui suppose que le fabricant de ce lecteur en ait eu connaissance, sans doute dans le cadre d'un « engagement de confidentialité ».

Mais il suffit d'intercaler le présent « espion » entre une carte Eurochip et un lecteur CyberMouse, pour voir clairement comment on présente le fameux « challenge », et même pour localiser de subtiles inversions de bits qui pourraient bien faire partie de la réponse de l'algorithme cryptographique !

Et n'oublions pas la possibilité d'étudier ce qui se passe quand on insère une carte exigeant une « haute tension » de programmation *Vpp* (par exemple une GPM 416 à ST1301) dans un lecteur qui ne la lui fournit pas (par exemple un CyberMouse).

On voit très bien, sur notre exemple, échouer la tentative d'écriture d'un bit dans le compteur de présentations de codes PIN erronés.

Il en irait tout autrement avec une carte équipée d'un circuit SLE 4404, un « équivalent » signé Infineon (Siemens), mais qui n'a pas besoin de *Vpp*. Prudence, donc !



L'espion complet, en ordre de marche.

CHAPITRE PAGE

4 AUTOUR DES CARTES SIM

Présente dans chaque téléphone portable GSM, la carte SIM fait partie des cartes à puce les plus répandues dans le public.

Nous avons montré, dans notre ouvrage *Téléphones portables et PC*, à quel point il peut être intéressant de l'explorer à l'aide d'un PC équipé d'un lecteur approprié, que cela plaise ou non à l'opérateur qui l'a émise !

Dans ce nouvel ouvrage, nous allons découvrir les mécanismes les plus intimes des cartes SIM les plus récentes, mais aussi de services dont on n'utilise bien souvent qu'une infime partie des possibilités (parfois cachées).

4.1 UNE BASICSIM « PHASE 2+ »

Nous avons montré, dans notre précédent ouvrage, comment transformer une BasicCard « Professional » en carte SIM permettant de « bricoler » librement avec la majorité des téléphones portables, même « verrouillés » par un opérateur.

Comme on pouvait s'y attendre, les versions de cartes se sont succédées à un rythme effréné, et la ZC 4.1 employée à l'époque est déjà complètement dépassée par la ZC 5.4 (et que dire de la toute récente ZC 5.5 ?).

Nous avons profité du « portage » de notre application « BasicSIM » sur ces nouvelles plates-formes matérielles, pour en exploiter les possibilités accrues.

Le dossier « SIM » du cédérom accompagnant cet ouvrage contient ainsi un fichier BasicSIM.IMG, qui attend avec impatience d'être chargé dans une ZC 5.4 « Rev. A » (distribuée en France par http://www.hitechtools.com).

Un sous-dossier est consacré à la ZC 5.5 (Rev. B), une toute nouvelle version offrant une capacité mémoire encore supérieure.

Différents fichiers batch facilitent au maximum l'opération, selon le modèle de lecteur de cartes installé sur le PC.

Tous trois mettent à contribution l'utilitaire BCLOAD.EXE, extrait du kit BasicCard ayant servi à la compilation de notre code source.

Grâce à la nouvelle « puce » dont est équipée la carte, il est dorénavant possible de l'utiliser dans certains téléphones à technologie « 3 volts », autorisés à interrompre le signal d'horloge de la SIM lorsqu'ils passent en mode « faible consommation » (ce qui « plantait » carrément le système d'exploitation des ZC 4.x).

Mais ce n'est pas tout, loin de là : cette nouvelle « BasicSIM » appartient à la « phase 2+ » et non plus à la « phase 2 », suppor-

tant par conséquent des fonctionnalités « SIM Toolkit » (il reste néanmoins possible de la « brider » à la phase 2, en remettant à 02h l'unique octet que contient le fichier 7F20:6FAE).

En pratique, elle dotera tout téléphone portable compatible, d'un « menu » supplémentaire, baptisé « Proactive ».

Parmi les fonctions « Proactive SIM » auxquelles il donne accès, signalons la lecture instantanée du « Terminal Profile » du téléphone, ainsi que l'activation, la désactivation, et le vidage du fichier « log » dans lequel la carte peut enregistrer les commandes qu'elle reçoit.

Le programme « SPYUTIL.EXE » offre des possibilités comparables, depuis le lecteur du PC, mais aussi et surtout celle de rapatrier ce fichier CARD.LOG sur le disque dur, à des fins d'analyse approfondie.

À noter que, contrairement à la version précédente, cette « BasicSIM » n'a pas besoin d'être initialisée avant sa première utilisation (elle le fait spontanément).

Bien entendu, nos lecteurs sont invités à se reporter à l'ouvrage précité (2e édition, août 2002) pour découvrir, si ce n'est déjà fait, les passionnantes expériences auxquelles un tel outil permet de se livrer, en toute liberté vis-à-vis des opérateurs.

4.2 EXPLORER LES CARTES « SIM TOOLKIT »

Tout comme notre nouvelle « BasicSIM », la plupart des cartes SIM récentes appartiennent à la « phase 2+ », et sont donc compatibles « SIM Toolkit » ou « Proactive SIM ».

Cela signifie non seulement qu'elles peuvent greffer leur propre menu sur celui du téléphone, mais aussi qu'elles sont capables d'émettre et de recevoir des appels ou des SMS à l'insu de l'utilisateur !

Cela mérite assurément toute notre vigilance, car des facturations inattendues peuvent en résulter, tandis qu'il n'est pas invraisemblable de craindre que ce genre de mécanisme ne puisse servir à des fins malveillantes, voire à la prolifération de virus d'un nouveau genre.

Cela étant précisé, il faut bien reconnaître que cette technologie très prometteuse permet aux opérateurs (ou à des prestataires tiers) de proposer des services extrêmement innovants, dont on ne pourra toutefois profiter que sur un téléphone raisonnablement récent.

Rien de plus simple que de savoir si une carte SIM est potentiellement compatible « STK » : il suffit de lire l'unique octet que contient son fichier « Phase » (7F20:6FAE).

S'il est supérieur à 02h, on pourra décoder le contenu de sa « SIM Service Table » (7F20:6F38) pour avoir (beaucoup) plus de détails.

Mais on peut aussi regarder comment la carte réagit à une commande « Terminal profile », par exemple `A0 10 00 00 04 FF FF FF FF` :

- si elle répond 6D00 (INS = 10h non reconnu), c'est qu'elle ne supporte pas le SIM Toolkit ;
- si elle répond 9000, cela prouve qu'elle supporte le SIM Toolkit, mais qu'elle n'a pas, du moins pour le moment, d'ordre à donner au téléphone ;
- si elle répond 91XX, elle est non seulement compatible SIM Toolkit, mais prête à transmettre des ordres au téléphone, sous la forme d'un bloc de données long de XXh octets.

Un logiciel de recherche de menus

Nous avons imaginé un petit programme ZCBasic permettant d'aller plus loin dans cette voie, en déterminant si la carte est en mesure d'insérer son propre menu dans celui du téléphone.

Le code source STKM.BAS a ainsi été compilé en STKM.EXE, une véritable application « console » pour Windows 95 ou supérieur, capable de se servir de n'importe quel lecteur PC/SC correctement installé.

Voici, par exemple, le résultat obtenu à partir de notre BasicSIM de « seconde génération » :

```
Proactive
Term. Profile
BasicCard
LOG On
LOG Off
Clear LOG
```

La première ligne correspond à l'entrée ajoutée par la carte dans le menu principal du téléphone, et les suivantes aux différentes rubriques qui peuplent ce nouveau « sous-menu ».

L'exemple ci-dessous provient, pour sa part, d'une carte « postpaid » émise par un opérateur français bien connu :

```
Services +
services
```

```
kiosque
SMS infos
agenda
```

On y retrouve quelques « raccourcis » vers des services offerts par le réseau, étant bien entendu que chaque choix peut, à son tour, donner accès à un nouveau sous-menu (que notre programme ne va pas jusqu'à détailler, car ce serait bien plus compliqué).

Ici, l'avantage du SIM Toolkit est d'uniformiser totalement le mode d'accès à ces services, quel que soit le modèle de téléphone utilisé (car on sait bien qu'il existe des différences considérables d'ergonomie d'une marque à l'autre).

Mais le revers de la médaille, c'est que ce surcroît de simplicité camoufle habilement le fait que les services en question fonctionnent souvent par envoi de SMS, évidemment payants, et même généralement surtaxés…

C'est particulièrement flagrant dans le cas de l'application « agenda », qui pourrait parfaitement être gérée localement dans le téléphone ou dans la carte (puisque celle-ci héberge déjà l'annuaire personnel de l'utilisateur).

Eh bien non ! Tout le contenu de l'agenda « SIM Toolkit » réside dans un serveur hébergé par l'opérateur…

Outre les problèmes de confidentialité et de sécurité que cela pose, il en résulte qu'un SMS (sinon plusieurs) est émis à chaque fois que l'utilisateur consulte ou actualise son agenda !

La mise en évidence irréfutable de ce genre de piège suppose des investigations assez poussées, par exemple au moyen d'un « espion » de cartes asynchrones tel que celui décrit au chapitre 2.

Elle nécessite également une bonne connaissance des modalités de dialogue entre une carte « SIM Toolkit » et un téléphone compatible, autrement dit de la spécification GSM 11.14 (téléchargeable sur http://www.etsi.org).

Comme l'appellation « Proactive SIM » le laisse entendre, une carte « Phase 2+ » peut prendre le contrôle du téléphone, alors que selon la norme ISO 7816, c'est pourtant le « terminal » qui est le maître absolu des échanges avec la carte.

Cette apparente contradiction est adroitement résolue par une extension des valeurs que peuvent prendre les octets de compte rendu SW1SW2, que retourne la carte après avoir exécuté une commande.

Si la carte répond 91XX au lieu de 9000 en cas d'exécution correcte, cela signifie qu'elle a un ordre long de XXh octets à transmettre au système d'exploitation du téléphone.

Celui-ci va alors spontanément le chercher, en envoyant une commande « FETCH » de la forme `A0 12 00 00 XX`.

En début de session, il s'agira souvent d'un ordre d'insertion d'une rubrique supplémentaire dans le menu d'origine du téléphone.

Comme on le voit dans notre code source STKM.BAS, un tel ordre commence par un octet D0h, suivi d'un octet précisant la longueur de ce qui vient après.

Prenons l'exemple d'un tel ordre « SET-UP MENU » qui commencerait par les octets suivants :

```
D0 30 81 03 01 25 00 82 02 81 82 85 09
```

Il s'agit, en pratique, d'une suite d'éléments « TLV » (*Tag*, *Length*, *Value*), dont le premier s'écrit : `81 03 01 25 00`.

Le « tag » 81h (nature de l'élément = « command details ») est suivi d'un octet de longueur (03h), puis très logiquement de trois octets de données.

Parmi ceux-ci, l'octet 25h identifie un ordre « SET-UP MENU ».

L'élément suivant se compose d'un tag 82h (nature = « identities »), d'un octet de longueur 02h, et de deux octets de données : 81h indiquant que les données qui vont suivre proviennent de la carte SIM, et 82h qu'elles sont destinées au système d'exploitation du téléphone.

Cela paraît évident, mais le destinataire pourrait tout aussi bien être le réseau (83h) en cas d'émission d'un SMS à l'initiative de la carte, l'écouteur (03h) pour l'émission d'un « bip », ou encore l'écran (02h) pour l'affichage d'un texte.

Suit un élément qui commence par un tag 85h (nature = « alpha ID ») et qui contient 09h octets de données (en l'occurrence le texte « Proactive » qui n'est autre que l'intitulé du menu à insérer).

Les rubriques de ce menu suivront enfin, annoncées chacune par un tag 8Fh (nature = service item), un octet de longueur, et un numéro de rubrique :

- `8F 0E 01` pour une rubrique N° 1 baptisée « Term. profile » ;
- `8F 0A 02` pour une rubrique N° 2 intitulée « BasicCard », etc.

Bien entendu, le téléphone rend compte à la carte de la bonne exécution (ou non !) de cet ordre, ce qui se fait par une commande « Terminal Response » (CLA = A0h, INS = 14h).

À partir du moment où le nouveau menu est installé, l'utilisateur peut en sélectionner une option à tout moment.

Dès que cela se produit, le téléphone prévient la carte en lui envoyant une commande « Menu Selection Envelope » (INS = C2h).

Dans notre cas, ce pourrait être :

```
A0 C2 00 00 09 D3 07 82 02 01 81 90 01 02
```

Le dernier octet stipule ainsi que c'est la rubrique N° 2 (BasicCard) qui a été sélectionnée.

On remarque, juste après les cinq octets d'en-tête de la commande, un tag D3h précisant que ce qui suit se compose encore d'éléments « TLV », dont la longueur totale est 07h octets.

Mais en pratique, c'est surtout le dernier de ceux-ci qui nous intéresse !

Une fois sélectionnée l'une des applications accessibles depuis le menu principal (SMS infos, par exemple), c'est par un processus différent (ordre « SELECT ITEM ») que la carte présentera, s'il y a lieu, les rubriques du sous-menu correspondant (se divertir, météo, bourse, etc.).

Cette fois, c'est par une commande « Terminal response » que le téléphone indiquera à la carte, le moment venu, la sous-rubrique choisie par l'utilisateur.

À ce stade, l'application « SIM Toolkit » sélectionnée se met immédiatement au travail, et risque d'avoir besoin d'un peu de temps avant de pouvoir retourner le compte rendu SW1SW2 (très probablement de la forme 91XX).

La carte prévient alors le téléphone par émission d'un ou plusieurs octets 60h (« More time »).

Cela peut vite devenir exaspérant avec certaines cartes basées sur la technologie « Java », pas particulièrement rapide…

Expertise de quelques anomalies

On aura compris que le langage STK, à la fois riche et compact, est passablement compliqué, au point même d'être assez médiocrement maîtrisé par certains développeurs d'applications travaillant pour les opérateurs ou les fabricants de téléphones portables.

Nous avons ainsi « expertisé » deux dysfonctionnements dont les « services clients » des opérateurs renvoient volontiers la

responsabilité sur les fabricants des téléphones, et ceux-ci… sur la carte SIM !

Premier exemple : une même carte SIM donne correctement accès aux informations météo par SMS sur un téléphone de marque A, mais pas du tout sur un appareil de marque B.

Le dépouillement des enregistrements effectués avec notre « espion » met en lumière des différences au niveau de la commande « Terminal response » envoyée à la carte, suite à la sélection de la rubrique « météo » (N° 13h) du menu « SMS infos » :

`A0 14 00 00 0F 81 03 01 24 00 82 02 82 81 03 01` *00* `10 01 13` (succès)

`A0 14 00 00 0F 81 03 01 24 00 82 02 82 81 83 01` *02* `90 01 13` (échec)

Certaines sont parfaitement légitimes, car elles correspondent à des variantes autorisées par la norme (03h au lieu de 83h, ou 10h au lieu de 90h).

En revanche, l'octet 02h au lieu de 00h signifiant « missing information » au lieu de « success », on peut excuser la carte de refuser de lancer l'application, bien qu'elle dispose en réalité de toutes les informations nécessaires.

Le téléphone est donc le principal coupable, mais la carte pourrait tout de même faire preuve d'un peu plus de souplesse…

Second exemple : l'application « météo » étant enfin ouverte, la carte demande à l'utilisateur d'entrer le numéro du département pour lequel il réclame des prévisions.

Seulement, le téléphone affiche des lettres majuscules alors que l'on veut taper des chiffres !

L'explication se situe dans le contenu de la commande « Fetch » par laquelle le téléphone va chercher son ordre de lecture du clavier :

```
A0 12 00 00 27 D0 25 01 03 01 23 01 02 02 81 82 0D 11 04
6E 6F 2E 20 64 05 70 74 20 20 20 20 20 20 20 20 11 02 02
02 17 03 04 50 4D
```

La carte donne la valeur 01h à l'octet « qualifier » précisant le format des données à recueillir, ce qui correspond à l'alphabet « par défaut » (et donc alphanumérique).

Tout se passe bien avec les téléphones qui commencent par afficher un chiffre, quitte à appuyer plusieurs fois sur la même touche quand on désire entrer une lettre.

Un problème se pose, par contre, avec les téléphones qui attendent du texte en priorité, ce qui est leur droit le plus strict dans cette situation.

Pour éviter ce genre de malentendu, il aurait suffi que la carte mette le « qualifier » à 00h, ce qui correspond à un alphabet purement numérique.

Cette fois, la faute incombe clairement à la carte, mais comme elle ne se manifeste que sur certains modèles de téléphones, le SAV de l'opérateur a tendance à incriminer le mobile lorsqu'il procède, en toute bonne foi, à ses inénarrables « essais croisés ».

Proactive SIM et envoi de SMS

Admettons maintenant que l'application « météo » soit enfin opérationnelle, et que l'on ait réussi à entrer le numéro de département 76.

Le véritable programme qui s'exécute dans la carte demande alors au téléphone d'envoyer un SMS à un service spécialisé.

Pour ce faire, la carte va, par exemple, répondre 9135 au lieu de 9000 dès qu'elle en aura l'occasion.

Le téléphone répliquera « à vos ordres » en émettant une commande « FETCH » réclamant tout juste 35h octets de données : `A0 12 00 00 35`.

En l'occurrence, ces 53 octets seront :

<pre>
D0 33 01 03 01 <u>13</u> 00 02 02 81 83 05 0C 6D 05 74 05 6F 20
20 20 20 20 20 20 06 07 91 <u>33 86 09 40 00 F0</u> 0B 11 01 00
05 81 <u>02 22 F0</u> 00 F6 07 6D 05 74 05 6F <u>37 36</u>
</pre>

Sans procéder à une analyse par trop approfondie de ce bloc de données, on notera que l'octet 13 (en sixième position) identifie un ordre « SEND SMS ».

Plus loin, un élément TLV avec un tag 05h (nature = « alpha ID », tout comme 85h), et une longueur de 0Ch octets, contient le texte destiné à s'afficher sur l'écran du téléphone : « météo », suivi de sept espaces (20h), soit un total de douze octets (0Ch).

La suite du bloc de données contient le corps du SMS à envoyer, et en particulier :

- le numéro du centre de messagerie chargé de l'acheminer (+33689004000) ;
- le numéro (surtaxé !) du destinataire (20220) ;
- le texte du message : météo76.

Rien de plus, en somme, que ce que l'on enverrait manuellement si on ne se faisait pas aider par une application « Proactive SIM » !

On remarquera, au passage, que l'alphabet utilisé diffère de l'ASCII au niveau des caractères accentués (le « é » étant codé 05h).

Imaginons enfin que l'envoi du SMS échoue (par exemple parce que l'on se trouve dans une zone non couverte par le réseau).

Le téléphone signalera cet échec à la carte au moyen d'une commande « Terminal response » du genre :

`A0 14 00 00 0D 01 03 01 13 00 02 02 82 81 03 02 20 00`

On y remarquera la répétition d'une partie de la commande précédente, à commencer par l'octet 13h (« SEND SMS »).

Les deux derniers octets précisent, quant à eux, la raison de l'échec (téléphone actuellement incapable de traiter la commande).

La carte réagira généralement en faisant s'afficher un message d'erreur approprié.

Grâce à un compte rendu 9119 au lieu de 9000, elle priera le téléphone de venir chercher 19h octets de données, ce qu'il fera au moyen d'une nouvelle commande « FETCH » : `A0 12 00 00 19`.

La carte transmettra alors les données suivantes :

```
D0 17 01 03 01 21 01 02 02 81 02 0D 0C 04 45 63 68 65 63
20 65 6E 76 6F 69
```

L'octet 21 en sixième position identifie un ordre « DISPLAY TEXT », ledit texte (« Echec envoi ») occupant les onze derniers octets du bloc de données.

On le voit, ce genre d'investigation n'est pas précisément simple, mais permet de mettre en évidence des quantités de choses que les opérateurs s'évertuent à dissimuler. Cela mérite bien quelques efforts, à commencer par la lecture attentive de la spécification GSM 11.14 !

4.3 « BRICOLER » LES SMS

Les SMS (*Short Message Service*) constituent un service essentiel du système GSM, et sont surtout connus sous la forme de « mini-messages » d'un maximum de 160 caractères affichables.

Il faut cependant savoir que la norme définit de très nombreuses variantes, dont des possibilités de « concaténation » permettant de mettre plusieurs messages bout à bout pour transporter des volumes de données plus importants.

Il existe aussi des catégories de SMS dont la réception peut se faire totalement à l'insu de l'utilisateur, en vue d'opérer des modifications plus ou moins profondes dans les cartes SIM ou les mémoires non volatiles des téléphones.

En milieu industriel, enfin, les SMS sont de plus en plus souvent utilisés pour télécommander et télésurveiller toutes sortes d'équipements munis de « modems GSM », y compris à bord des trains.

Les applications dites *Over The Air* (OTA) des SMS sont à cent lieues des « petits mots » que s'échangent si volontiers les jeunes (et parfois les moins jeunes !), en mode « texte ».

C'est évidemment par SMS que certains prestataires peuvent télécharger à distance (et bien entendu à titre payant) les répertoires de numéros des cartes SIM de leurs clients, des mélodies de sonnerie, ou des logos graphiques.

C'est aussi par SMS que s'est fait le basculement d'Itinéris à Orange, astucieusement effectué par remplissage à distance du fichier 7F20:6F46 des cartes SIM.

Écrite dans ce fichier « Service Provider Name », la chaîne d'octets `00 4F 72 61 6E 67 65 20 46 FF FF FF FF FF FF FF FF` impose (du moins sur les mobiles compatibles) l'affichage du texte « Orange F » à la place du nom d'origine du réseau sur lequel le mobile est inscrit, sauf à l'étranger où les deux peuvent même apparaître ensemble.

Grâce à la technologie « ESMS » si chère à Gemplus, ce genre de manœuvre fonctionne même avec certaines cartes SIM de phase 2.

En gros, le SMS contient une suite de commandes, que la carte est priée d'exécuter comme si elle les avait reçues directement du terminal dans lequel elle est insérée.

Cela peut aller jusqu'à la création de nouveaux fichiers, grâce à des commandes « administratives » qui diffèrent, hélas, d'un fabricant de cartes à l'autre.

Les choses peuvent aller infiniment plus loin avec les cartes SIM de Phase 2+, autrement dit « SIM Toolkit », qui adhèrent cette fois à une normalisation très précise.

Non contentes de pouvoir envoyer spontanément des SMS, comme nous venons de le voir, elles peuvent aussi en recevoir très discrètement.

Cela aussi bien pour les besoins d'une application résidant déjà dans la carte (paiement électronique, par exemple), que pour la mise en place de telle ou telle fonctionnalité nouvelle.

Dans un contexte « multiapplication », la responsabilité de la manœuvre incombe à l'opérateur ayant émis la carte, même s'il l'effectue pour le compte d'un tiers (par exemple, une banque).

La plupart du temps, elle se traduit par l'apparition de nouvelles options dans le système de « menus » du téléphone, mais une application peut aussi rester parfaitement cachée, voire « dormante ».

Bien évidemment, de telles possibilités de téléchargement de code exécutable (souvent des « applets » Java) peuvent faire craindre l'introduction de virus, éventuellement susceptibles de se propager (toujours par SMS) aux numéros enregistrés dans le répertoire de la carte SIM.

Sans les spécifications GSM 02.48 et 03.48, qui préconisent des mesures de sécurité… parfaitement facultatives, on pourrait même craindre que n'importe qui puisse, par exemple, inscrire des numéros malveillants dans les annuaires des cartes SIM, ou expédier de fausses factures vers des mobiles équipés de fonctions de paiement.

Cela nécessiterait tout de même un peu d'astuce, car un téléphone portable normalement constitué ne permet pas de composer autre chose que des SMS très « ordinaires ».

Un logiciel de création de SMS

Qu'à cela ne tienne ! Notre programme SMS7.BAS (dossier « SMS » du cédérom) prouve amplement que quelques lignes de Basic suffisent pour développer un outil capable de construire de toutes pièces, dans une carte SIM valide, un SMS prêt à être expédié :

Toujours écrit en ZCBasic, il pourra être personnalisé selon les besoins de chacun, puis compilé à l'aide du kit BasicCard (version 4.32 ou supérieure).

Pourvu que le PC soit équipé d'un lecteur de cartes à puce PC/SC (ou CyberMouse, à condition de modifier la valeur de *ComPort* en conséquence), il enregistrera un SMS « sur mesures » en première position dans la carte SIM que l'on voudra bien y insérer.

Auparavant, on aura désactivé le code PIN de la carte (cela se fait à partir du menu du téléphone), et vérifié que l'actuel SMS N° 1 peut être « écrasé » sans regrets (les SMS de rang supérieur ne seront aucunement affectés).

Chaque champ potentiellement intéressant à « bricoler » se présente sous la forme d'une chaîne séparée :

- **SC$** contient, en format téléphonique international, le numéro du SMSC (centre de messagerie) que l'on veut charger d'acheminer le SMS. En règle générale, ce sera celui de l'opérateur ayant émis la carte SIM (par exemple 33689004000 pour Orange, 33609001390 pour SFR, 33660003000 pour Bouygues).
  Si la formule souscrite autorise l'accès à des SMSC étrangers, alors il n'y a plus de limites : on pourra aussi bien choisir de transiter par la Suisse (41794999000), l'Afrique du Sud (27830000410), Singapour (6596500005), etc. (On trouve sur Internet des listes entières de SMSC, dont certains sont également accessibles par modem !)
  Le principal intérêt de la chose est que tous les SMSC n'appliquent pas nécessairement les mêmes règles de rejet des SMS « non-conformistes », mais attention ! Certains opérateurs (et notamment les trois français) refusent les SMS provenant des SMSC de certains pays. Ceci explique peut-être cela…

- **To$** contient, toujours en format international (mais sans le signe +), le numéro du destinataire du SMS (en général le numéro d'un mobile dont on est le légitime propriétaire, ou avec lequel on est dûment autorisé à procéder à de tels essais).

- **SR$** aura la valeur 11h si l'on ne réclame pas d'accusé de réception, ou 31h dans le cas contraire.

- **PID$** définit le « protocole » que l'on souhaite employer, la valeur par défaut qu'utilisent les téléphones portables étant 00h.
  La valeur 5Fh suggère ici au mobile d'offrir une possibilité de rappel immédiat du numéro de l'expéditeur : une formule normalement destinée aux opérateurs (pour avertir de l'arrivée d'un message dans le répondeur), mais que l'on peut très facilement s'approprier !
  Les valeurs 7Fh, 7Dh, et 7Eh identifieraient des SMS à destination, respectivement, du système d'exploitation de la carte SIM ou du téléphone, ou destinés à déverrouiller le mobile à distance.
  Il est intéressant d'observer dans quelle mesure les réseaux et/ou les SMSC refusent l'acheminement des SMS de cet acabit, potentiellement suspects lorsqu'ils émanent d'un particulier et non d'un opérateur.
  Une valeur 3Fh, enfin, indiquerait au SMSC qu'on lui demande de se charger d'une conversion de format, par exemple de 8 bits vers 7 bits.

- **DCS$** décrit, précisément, le type de format utilisé, la valeur par défaut étant là encore 00h. Dans ce mode « texte », on utilise un codage affreusement compliqué, servant juste à « emballer » 160 caractères à 7 bits dans 140 octets (ce qui n'est pourtant pas

un exploit !). La valeur F0h correspond ainsi au codage sur 7 bits d'un SMS de « classe 0 », ne devant pas être enregistré dans la carte SIM lors de sa réception, mais seulement affiché (Message « Flash »). 04h indiquerait par contre que le message, codé sur 8 bits, n'appartient à aucune « classe » particulière et doit donc être automatiquement sauvegardé.
C'est là que l'on butera sur une caractéristique de certains téléphones récents, qui refusent de transmettre (et/ou d'afficher) les SMS appliquant ce codage « utilisateur » à 8 bits. Les mobiles les plus anciens ne souffrent généralement pas de cette limitation, et s'achètent d'occasion pour une bouchée de pain. Un modèle convenant particulièrement bien aux expériences qui nous intéressent est le Philips FIZZ (information « libre de toute publicité » !).

- **VAL$** précise la durée de validité du message, c'est-à-dire le temps pendant lequel le SMSC est prié de tenter d'acheminer le message si le mobile destinataire est indisponible (par exemple arrêté). Cette valeur obéit à un codage complexe, mais voici quelques exemples amplement suffisants : 05h (30 minutes), 0Bh (1 heure), 17h (2 heures), A7h (24 heures), ABh (5 jours), FCh (60 semaines !).

- **TXT$** représente le contenu utile du message, dont LTX$ indique la longueur en caractères. En codage 8 bits, ce sera tout simplement le nombre d'octets, mais en codage 7 bits, ce sera le nombre de septets, supérieur ou égal, donc, à la longueur de TXT$.
Dans notre exemple, TXT$ contient le texte « Bonjour ! » codé sur 7 bits, mais notre programme SMS8.BAS exploite une variante permettant d'opérer « en clair » sur 8 bits, à charge pour le SMSC d'opérer la conversion en 7 bits :

```
PID$=Chr$(&H3F)
DCS$=Chr$(&H04)
TXT$="Bien le bonjour !"
LTX$=Chr$(Len(TXT$))
```

Il est intéressant d'observer ce que devient un même message, selon qu'on le transmet avec tel ou tel modèle de téléphone, ou par l'entremise de tel ou tel opérateur français ou étranger.

Voila ainsi ce que donne notre exemple, confié à un Philips FIZZ équipé d'une carte SIM émise par Orange avec le numéro +33 6 8a bc de fg (SMSC +33 6 89 00 40 00) :

```
07 07 91 33 86 09 40 00 F0 11 FF 0B 91 14 97 ih
kj ml Fn 3F 04 17 11 42 69 65 6E 20 6C 65 20 62
6F 6E 6A 6F 75 72 20 21 FF ................. FF
```

La transmission a été effectuée (le 22/11/02 à 17h 30' 54") vers un Motorola M3188, inscrit sur le réseau Bouygues, mais équipé d'une carte SIM suisse (numéro de la forme +41 79 hi jk lmn), dans laquelle il a été retrouvé ceci :

```
01 07 91 33 86 09 40 00 F0 04 0B 91 33 86 ba dc
fe Fg 3F 00 20 11 22 71 03 45 00 11 C2 74 D9 0D
62 97 41 E2 B7 5B FD AE CB 41 21 FF ........ FF
```

On constate que la conversion de 8 bits en 7 bits a bien été effectuée, mais que l'indicateur de protocole est resté à 3Fh (il aurait tout aussi bien pu être remis à 00h par le SMSC).

Mais certains téléphones se permettent de « rafraîchir » le message dans la carte SIM avant de l'envoyer, redemandant même parfois à cette occasion un numéro de destinataire pourtant déjà présent.

Il n'est alors pas rare que PID$ et DCS$ soient rétablis tous deux à leur valeur par défaut de 00h.

En pareil cas, le SMSC n'effectue évidemment pas de conversion, d'où soit un affichage inintelligible, soit un avertissement du genre « Format de message incompatible » ou « Données 8 bits ».

Là encore, ce sont les modèles récents qui paraissent les plus enclins à ce genre de « censure ».

Il est intéressant de procéder à des essais « à blanc », sans transmission effective (et donc gratuitement), en utilisant une « Basic-SIM ».

Donnant accès à toutes les fonctions du téléphone dans lequel on l'insère, mais ne s'inscrivant sur aucun réseau, celle-ci permet de simuler la transmission d'un SMS, puis de relire celui-ci à l'aide de tout bon éditeur de carte SIM. C'est à ce moment que l'on se rendra compte de ce que le téléphone s'est permis ou non de modifier subrepticement.

Pour aller (beaucoup) plus loin dans cette passionnante exploration, on ne saurait trop conseiller la lecture attentive des spécifications GSM 03.40 et GSM 03.38. Rappelons qu'elles sont de nature publique, et téléchargeables gratuitement sur http://www.etsi.org.

Des paiements électroniques par SMS

Le paiement par téléphone portable « bi-fente », équipé d'un lecteur de carte bancaire, est basé sur une carte SIM « phase 2+ », hébergeant une application « SIM Toolkit » spécialisée.

Ce véritable logiciel embarqué communique, d'une part, avec l'utilisateur par l'afficheur et le clavier, et d'autre part, avec le serveur monétique par échange de SMS… pas forcément cryptés.

Normalement émis et reçus sans que l'utilisateur ne s'en rende compte, ceux-ci peuvent être détournés de leur itinéraire normal, moyennant un peu d'astuce.

L'exemple suivant est une « facturette » reçue par un mobile, en réponse à une demande de rechargement de sa formule prépayée. À ce stade, il ne s'agit que d'une opération « *pro-forma* », que l'on peut valider (en l'authentifiant avec une carte bancaire) ou abandonner.

Le mobile ayant initié la procédure de rechargement a été arrêté juste après la sélection du montant de la recharge, puis sa carte SIM a été retirée.

Insérée immédiatement dans un autre téléphone, suffisamment vieux pour ne pas supporter le « SIM Toolkit », celle-ci a accueilli la « facturette » comme un SMS ordinaire, on ne peut plus facile à examiner avec des outils courants.

En effet, si le SMS en question se trouve en première position dans la carte SIM, il suffit d'enchaîner les trois commandes suivantes pour en afficher le contenu, par exemple avec SCARD.EXE :

`A0 A4 00 00 02 7F 10` (avec Le = 00)
`A0 A4 00 00 02 6F 3C` (avec Le = 00)
`A0 B2 01 04 B0` (avec Lc = 00)

Transféré par « coupé-collé » vers un simple éditeur de texte (« bloc-notes » de Windows), le fichier ainsi prélevé sera alors sauvegardé, puis soumis à un éditeur hexadécimal tel que PDED.EXE (disponible après installation du logiciel de démonstration fourni dans le répertoire « MQP » du dossier « SMS »).

```
03 07 91 33 86 09 40 00 F0 04 05 83 12 00 F0 7F   ··æ3å·@····â
F6 20 11 71 71 81 31 00 8C 01 35 14 00 35 20 00   ·  ·qqü1·î·5··5
00 25 00 09 78 17 18 11 11 17 02 68 52 10 30 36   ·%··x······hR·06
38 33 3x 3x 3x 3x 3x 3x 30 34 00 00 00 01 82 06   83xxxxxx04····é
2A 34 32 38 37 30 36 3x 3x 3x 30 30 30 37 31 08   *428706xxx00071
18 44 66 6C 61 20 6D 6F 62 69 63 61 72 74 65 05   ·Dfla mobicarte
81 12 00 F0 FF FF FF FF FF FF FF 45 55 52 27 31   ü··········EUR'1
72 65 63 68 2E 32 35 1B 65 0A 2B 35 1B 65 20 6F   rech.25·e·+5·e o
66 66 65 72 74 73 0A 76 61 6C 69 64 69 74 05 3A   fferts·validité:
33 20 6D 6F 69 73 20 20 20 20 20 20 20 20 20 20   3 mois
20 20 20 20 FF FF FF FF FF FF FF FF FF FF FF FF
```

Le premier des 176 octets qu'occupe le SMS dans le fichier 7F10:6F3C de la carte SIM indique que nous sommes en présence d'un message reçu, non lu (il serait à 01h si on avait pris la peine de le faire défiler entièrement sur l'afficheur du mobile).

Suit le numéro du SMSC (centre de messagerie) ayant acheminé le SMS, l'octet de type (91h) indiquant qu'il est présenté en format téléphonique international (+33 6 89 00 40 00).

Le numéro de l'expéditeur du message (21000) est spécifique à l'opérateur (type 83h), et d'ailleurs accessible gratuitement, même en cas de crédit épuisé (sinon, on ne pourrait pas recharger !).

Les octets 7Fh et F6h précisent que le SMS était destiné à la carte SIM, et qu'il est codé en format « 8 bits ».

C'est ce qui explique que les textes (que l'application SIM Toolkit devait afficher sur l'écran du mobile) apparaissent « en clair » dans notre éditeur hexadécimal.

Les sept octets qui suivent indiquent la date (17/11/02) et l'heure (17h 18' 13") d'envoi du SMS, ainsi que le fuseau horaire local (ici 00h).

L'octet 8Ch, enfin, correspond à la longueur (140 octets) du contenu utile du message, qui le suit d'ailleurs immédiatement.

Celui-ci comprend (toujours en clair !), le montant à payer (25 €), la date et l'heure d'établissement de la facturette, le numéro du téléphone à recharger, et ce qui ressemble fort à une référence de compte à créditer ou à un « certificat ».

Pour des raisons évidentes, quelques chiffres y ont donc été occultés.

Des SMS malveillants ?

Parmi les SMS « non sollicités » que l'on reçoit de plus en plus fréquemment, il arrive aussi de trouver ce genre de « script », en provenance d'un numéro parfaitement inconnu :

```
//SCKE2 BEGIN:VCARD
N:Adeline
TEL:0610xxyyzz
END:VCARD
```

Le plus curieux est qu'en l'occurrence, l'utilisateur du mobile dont est censé émaner ce message ne voit pas mieux que le destinataire de quoi il peut bien s'agir : serait-il donc techniquement possible de se faire passer pour quelqu'un d'autre ?

En fait, nous sommes en présence d'une « carte de visite virtuelle » au format vCard, comme on en utilise couramment dans les e-mails.

Selon la spécification « *Smart messaging* » de Nokia, l'en-tête `//SCKE2` indique à tout téléphone compatible qu'il doit détourner ce SMS vers son port N° E2h, justement baptisé « *Business card exchange* ».

De quoi se retrouver (sait-on jamais ?) avec une entrée indésirable dans son annuaire personnel…

Seulement, le numéro de téléphone 06 10 xx yy zz était ici… celui auquel le SMS est parvenu !

Fausse manœuvre involontaire, ou acte de malveillance ? Toujours est-il que, reçu fort à propos par un téléphone non compatible, ce message s'est trouvé enregistré dans la carte SIM comme un SMS « ordinaire », prêt à subir une analyse approfondie…

Télécharger des sonneries

Le téléchargement de mélodies de sonnerie sur les téléphones portables est une véritable « poule aux oeufs d'or », pourtant basée sur un mode de transmission fort peu coûteux (encore et toujours les SMS !).

Moyennant un peu d'imagination, il suffit d'un mobile GSM et d'un PC muni d'un lecteur de cartes à puce, pour se lancer (à petite échelle !) dans cette activité très à la mode.

C'est à Nokia que l'on doit le « *Smart messaging* », un procédé permettant d'encapsuler toutes sortes de contenus évolués dans des mini-messages astucieusement structurés (voir http://www.forum.nokia.com).

Depuis, les organismes de normalisation (http://www.3gpp.org) ont imaginé une alternative publique à ce système « propriétaire », l'EMS (*Enhanced Message Service*).

Adopté par des marques concurrentes telles que Alcatel, Ericsson, Siemens, ou Motorola, il présente certains avantages, mais aussi quelques inconvénients, par rapport à son aîné entièrement incompatible.

Chacun comprendra donc pourquoi, avant de se faire envoyer (à prix d'or !) une sonnerie ou un logo, il faut indiquer d'une façon ou d'une autre la marque, voire le modèle, de son téléphone.

Reçue sur un mobile non compatible, une mélodie prendra souvent la forme d'un texte sybillin, mais dont l'analyse octet par octet sera, là encore, riche d'enseignements !

Tout aussi intéressante est l'étude du contenu de messages composés, mettons, sur un Alcatel One Touch 311 (compatible EMS), dans lesquels on aura inséré une mélodie.

Récupérons donc, sans même le transmettre, un SMS ainsi construit, puis soigneusement sauvegardé dans la carte SIM :

```
07 07 91 33 86 09 40 00 F0 41 FF 0B 91 33 x6 xx   ··æ3å·@··A··æ3
xx xx Fx 00 00 5F 4F 0C 4D 03 42 45 47 49 4E 3A   ..···_O·M·BEGIN:
49 4D 45 4C 4F 44 59 0D 0A 56 45 52 53 49 4F 4E   IMELODY··VERSION
3A 31 2E 30 0D 0A 46 4F 52 4D 41 54 3A 43 4C 41   :1.0··FORMAT:CLA
53 53 31 2E 30 0D 0A 4D 45 4C 4F 44 59 3A 64 33   SS1.0··MELODY:d3
65 34 64 34 65 33 3A 0D 0A 45 4E 44 3A 49 4D 45   e4d4e3:··END:IME
4C 4F 44 59 0D 0A 10 14 0E 01 FF FF FF FF FF FF   LODY············
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ················
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ················
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ················
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ················
```

Le point de départ de l'exploration se situe, ici, au niveau du dixième des 176 octets qu'occupe, dans la SIM, tout SMS normalement constitué : 41h.

Le fait que le bit de poids 64 y soit à 1 signale que ce message contient un UDH ou *User Data Header*.

En clair, cela signifie que le texte du message (`10 14 0E 01`, soit « ABC », codé en format « par défaut ») est précédé d'un bloc de données à vocation particulière, ne devant normalement pas être affiché (mais on a parfois des surprises !).

La longueur de cet « en-tête » est précisée, ici, dans le 23e octet : 4Fh, c'est-à-dire 79 octets.

Le premier de ceux-ci (0Ch) définit le type de contenu que renferme l'en-tête, selon les conventions de la spécification 3GPP TS 23.040 (version 6.0.1 de septembre 2002).

Suit à nouveau un octet de longueur (4Dh, soit très logiquement deux unités de moins), puis un 03h demandant que la mélodie soit jouée après l'affichage du troisième (et dernier) caractère du texte.

0Ch signifie, en effet, que nous sommes en présence d'un *user defined sound*, autrement dit d'une mélodie composée par l'utilisateur (ou prélevée dans l'échantillonnage offert d'origine avec le téléphone).

0Bh désignerait un son prédéfini (qu'un seul octet suffirait à identifier), 0Dh une animation prédéfinie, 10h un logo « bitmap » de 128 octets, etc.

Notons que cet IEI (*Information Element Identifier*) peut prendre 256 valeurs différentes, dont certaines sont des plus « sensibles » (contenus destinés à des interventions « *Over The Air* » dans la carte SIM, par exemple !).

En ce qui nous concerne, le contenu utile de l'en-tête est fort anodin : un texte, en ASCII « 8 bits », décrivant une courte partition musicale selon le codage « iMelody ».

Développé par l'association IRDA (http://www.irda.org/standards/pubs/imelody.pdf), ce format de description de mélodies a l'avantage de se présenter sous une forme directement lisible :

```
BEGIN:IMELODY
VERSION:1.0
FORMAT:CLASS1.0
MELODY:d3e4d4e3:
END:IMELODY
```

On peut trouver d'innombrables fichiers iMelody (portant l'extension .IMY) sur Internet, en obtenir par transcodage depuis d'autres formats musicaux (RTTTL, MIDI, etc.), ou bien (et c'est de loin le plus motivant !) en écrire de toutes pièces si l'on possède quelques notions de solfège.

Le format iMelody permet, en effet, de transcrire les caractéristiques essentielles d'une partition, que sont la hauteur et la durée des notes.

Dans le cas le plus simple, une note sera codée sous la forme d'une lettre (minuscule) suivie d'un chiffre.

Selon une correspondance familière aux musiciens, `c`, `d`, `e`, `f`, `g`, `a` et `b` représentent Do, Ré, Mi, Fa, Sol, La et Si.

On peut ajouter un bémol (`&`) ou un dièse (`#`) devant une note, tandis qu'il est obligatoire de faire suivre chaque note d'un chiffre précisant sa durée.

0 correspondant à une note « entière », 1 indique une durée moitié moindre, 2 une durée quatre fois moindre, et ainsi de suite jusqu'à 5 qui correspond à 32 fois moins (on retrouve là les notions classiques de « rondes », « noires », « croches », etc.).

Pour plus de finesse, on peut faire suivre le chiffre d'un signe « modificateur » de durée : un point pour une note pointée, ou un signe « deux points » pour une note double pointée.

Sauf indication contraire, les notes sont réputées appartenir à l'octave 4 (La à 880 Hz), identifiée par la notation « `*4` ».

S'il faut changer d'octave, on insérera l'indication correspondante (par exemple `*5`) avant la note à jouer.

Les octaves définies vont de `*0` (La = 55 Hz) à `*5` (La =1 760 Hz), mais les plus graves s'avèrent inexploitables sur les haut-parleurs de la plupart des téléphones portables !

De multiples possibilités supplémentaires sont offertes par la spécification iMelody, aux dépens toutefois du nombre de notes pouvant tenir dans les 140 octets de « charge utile » d'un SMS : tempo, style (normal, staccato, legato), volume, etc.

Attention ! Il faudra scinder toute ligne de texte iMelody plus longue que 75 octets.

Mais voyons maintenant comment transmettre soi-même une telle composition musicale !

Plusieurs moyens sont disponibles pour envoyer des SMS.

L'un des moins connus consiste à passer par un modem relié au réseau téléphonique fixe, ou par un cordon transformant un téléphone portable en « modem GSM ».

Des logiciels spécialisés, dont l'excellent SMS-it (http://www.mawnet.com), se chargent absolument de tout, au point même de provoquer une certaine frustration technique !

Pour notre part, nous avons imaginé de charger de nouveau le SMS directement dans une carte SIM valide, puis de le faire transmettre par le téléphone portable (très ordinaire) dont elle provient.

Cette approche se traduit par le petit programme EMSGOD.BAS, développé encore et toujours en ZCBasic.

Prévu pour être utilisé avec n'importe quel lecteur de cartes à puce « PC/SC » convenablement installé (`ComPort=101`), il pourrait tout aussi bien fonctionner, sans aucun driver, avec un CyberMouse ou un ACR 30 série (`ComPort=1` ou `ComPort=2`, selon le port COM utilisé).

Avant de retirer la carte SIM de son téléphone, on pensera à désactiver son code PIN, de façon à autoriser notre programme à en prendre le contrôle.

Son code source, écrit au plus court, devra être personnalisé et compilé avant chaque transmission, mais on pourrait facilement l'enrichir de « dialogues » permettant d'entrer, sans nécessité de recompilation, les éléments variables que sont, comme nous l'avons déjà vu :

- **SC$** : le numéro du serveur chargé d'acheminer le SMS ;

- **To$** : le numéro du destinataire ;
- la mélodie (c'est-à-dire le texte qui suit le mot-clef `MELODY:`).

Le programme s'étant exécuté avec succès, on doit retrouver le SMS à transmettre en position N° 1 de la carte SIM, avec en guise de texte un seul et unique caractère « @ ».

La durée de validité du message est arbitrairement fixée à 24 heures (A7h), mais pourrait éventuellement être allongée (ABh = 5 jours, par exemple).

Notons bien que l'on peut rencontrer des téléphones qui modifient abusivement les SMS qu'ils n'ont pas composés eux-mêmes ! Mais comme la plupart des GSM de cette marque, l'Alcatel One Touch Club qui a servi à nos essais ne se permet pas ce genre de fantaisie.

Les choses sont fort différentes en ce qui concerne les téléphones de marque Nokia.

En « *Smart messaging* », on fait appel à un codage plus compliqué, mais aussi plus compact, car défini bit par bit et non plus octet par octet.

Ainsi, trois octets 26h 82h A8h représenteraient deux notes (Ré et Mi), assorties chacune de leur durée (avec modificateur).

En iMelody V.1.0, il faudrait entre quatre et huit octets pour coder les mêmes informations.

Cette opportunité de transmettre des mélodies plus longues se paie toutefois par la quasi impossibilité de les composer manuellement.

Voici, par exemple, un SMS contenant une mélodie nettement plus longue que notre échantillon codé en iMelody : largement plus compact, mais à peu près indéchiffrable !

```
07 07 91 33 86 09 40 00 F0 51 FF 0B 91 33 x6 xx   ··æ3å·@··Q··æ3··
xx xx Fx 00 F5 A7 34 06 05 04 15 81 00 00 02 4A   ······4····ü···J
3A 59 05 B9 D1 A1 95 B4 04 00 37 1E 86 57 41 84   :Y···íò···7·åWAä
18 41 A4 17 49 86 1A 41 C4 1C 42 48 A1 07 12 69   ·Añ·Iå·A··BHí··i
86 10 69 06 10 5D 06 10 00 00 00 FF FF FF FF FF   å·i··]··········
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ················
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ················
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ················
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ················
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ················
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ················
```

À défaut d'un logiciel spécialisé, on peut songer à opérer par conversion depuis un autre format. Cela peut même se faire en

ligne, et gratuitement dans la limite de trois mélodies par 24 heures (http://www.convert-ringtones.com).

Une autre particularité du format Nokia est que le SMS est d'un type spécial (DCS = F5h au lieu de 00h). Cela implique qu'à la réception, il sera interprété « au vol », sans transiter par la carte SIM.

Normalement, il n'est donc pas question de le retransmettre « en douce » vers un autre mobile, ni même de procéder à son inspection, sauf si l'on s'arrange pour le recevoir sur un téléphone non compatible !

En fait, l'en-tête du SMS ne contient plus la mélodie elle-même : sa longueur n'est que de six octets, tandis que son type de contenu est 05h, ce qui correspond à des « numéros de ports » codés cette fois sur 16 bits : 1581h (destinataire) et 0000h (provenance).

Or, le port 1581h correspond, dans les téléphones Nokia, au « *Ringing Tone Reader* », vers lequel tout le reste du message sera automatiquement dirigé.

Celui-ci contient justement la mélodie, qui se termine par un « marqueur de fin » composé d'un bloc de bits à 0.

Malgré toutes ces complications, notre programme peut être transformé pour fonctionner en mode « *Smart messaging* » : NOKGOD.BAS.

La principale restriction est qu'on ne pourra pas écouter, en local, la mélodie sans la transmettre, alors que c'est très faisable avec un message « EMS ».

Dans N$, on retrouvera (en format « noktxt ») la mélodie contenue dans le SMS que nous venons d'analyser, la même d'ailleurs que celle codée en iMelody dans EMSGOD.BAS.

Ces deux variantes ont été converties automatiquement depuis une version MIDI du « *God Save the Queen* » trouvée sur un site… britannique.

Les « fausses notes » (en fait des sauts d'octave) que les plus musiciens de nos lecteurs ne manqueront pas d'y détecter, en disent long sur la difficulté de ce genre de transcodage, qu'il sera souvent nécessaire de fignoler manuellement.

Histoire de se fixer les idées, nous avons soumis une très simple mélodie (en fait, une gamme) à un service de conversion en ligne.

Parmi ses douze formats de sortie, dont certains hautement « propriétaires », voici quelques exemples assez représentatifs

des différentes approches que l'on peut suivre pour coder une partition.

Cela avec plus ou moins de bonheur, d'ailleurs, car les informations de rythme (durée ou tempo) ne sont pas toujours prises en compte.

Dans le format iMelody, à l'inverse, apparaissent les valeurs « par défaut » de paramètres que l'on a généralement tendance à omettre, afin de ne pas trop alourdir le message.

Le tempo, en particulier (mot clef `BEAT:`), peut varier entre 25 et 900 bpm, avec une valeur par défaut de 120 à la minute.

Il est intéressant de comparer les différentes notations utilisées pour indiquer les changements d'octave, point délicat s'il en est, tous les formats n'utilisant pas forcément les mêmes valeurs par défaut ni la même syntaxe.

Format iMelody (.imy)

```
BEGIN:IMELODY
VERSION:1.0
FORMAT:CLASS1.0
NAME:Gamme
COMPOSER:Unknown
BEAT:160
STYLE:S1
VOLUME:V7
MELODY:c4d4e4f4g4a4b4*5c4
END:IMELODY
```

Format RTTTL

```
Gamme:d=4,o=5,b=160:c,d,e,f,g,a,b,c6
```

Format Nokia Composer

```
c,d,e,f,g,a,b,c6
```

Format Nokia « texte binaire » (.noktxt)

```
024A3A551D85B5B5940400232086D7411413415416418 41A41C41140000000
```

Format Ericsson EMelody (.emy)

```
cdefgab+c
```

Format « touches clavier » Motorola V60

```
11222030330333040202201112220
```

Format « touches clavier » Samsung

```
88188288388488588688 7881
```

4.4 DES EXPÉRIENCES D'AUTHENTIFICATION

On sait bien que toute carte SIM est capable de s'authentifier auprès de l'opérateur qui l'a émise, et cela au travers de son propre réseau ou de ceux de ses partenaires étrangers.

Il nous a cependant paru utile de doter le cédérom de ce livre d'un petit programme ZCBasic (AUTHSIM.EXE) capable de faire jouer ce mécanisme à partir d'un simple lecteur PC/SC : on le trouvera dans le répertoire « AUTH » du dossier « SIM ».

Il est intéressant de le mettre en œuvre sur notre BasicSIM, dans la mesure où le résultat obtenu devrait rassurer tous ceux qui craignent (et on se demande bien pourquoi !) que celle-ci ne puisse servir à des fins inavouables :

```
A  U  T  H  E  N  T  I  F  I  C  A  T  I  O  N
41 55 54 48 45 4E 54 49 46 49 43 41 54 49 4F 4E
45 4E 54 49 46 49 43 41 54 49 4F 4E
```

Au lieu d'appliquer, au bloc de 16 octets qui lui est soumis, l'algorithme « A3A8 » (au demeurant secret !) que renferment les « vraies » cartes SIM, la nôtre retourne « bêtement » les 12 derniers.

Si l'on osait enregistrer l'identifiant (IMSI) d'une carte valide dans la BasicSIM, donc, un processus d'inscription pourrait effectivement être initié sur tel ou tel réseau français ou étranger, mais toute demande d'authentification échouerait forcément, n'est-ce pas ?

Pas question, par conséquent, de téléphoner gratuitement avec ce genre de carte, que l'on ne peut donc en aucune façon assimiler à un « clone ».

Cela méritait, nous semble-t-il, d'être fermement précisé…

Bien entendu, le même programme pourra être essayé avec de véritables cartes SIM, valides ou non, afin d'examiner le genre de réponse que l'on obtient, quitte à faire varier quelque peu les données envoyées.

De quoi procéder à d'édifiantes comparaisons avec les autres cartes auxquelles nous allons nous intéresser dans les prochains chapitres…

CHAPITRE PAGE

5 AUTOUR DES CARTES « SANTÉ »

Un peu partout dans le monde, il se dessine une tendance à la simplification des procédures bureaucratiques, grâce à des outils informatiques au premier rang desquels figurent les cartes à puce.

C'est particulièrement net en matière de cartes d'assurance maladie, voire de « cartes santé » au sens le plus large du terme.

Mais ce n'est qu'une étape sur le chemin d'un véritable « e-gouvernement » : il faut s'attendre, semble-t-il, à ce que chacun possède un jour une carte « multiapplication » qui remplacera, à elle seule, la carte nationale d'identité, la carte d'assuré social, le permis de conduire, la carte d'électeur, voire certains moyens de paiement et de signature électronique.

Cela soulève très naturellement des interrogations en matière de sûreté et de protection de la vie privée, que seule une franche transparence pourrait vraiment dissiper.

Or, on en est très loin : le « culte du secret » a la vie dure, même s'il a été amplement démontré que, loin de contribuer à la sécurité des applications les plus « sensibles », il excite fort logiquement la curiosité.

C'est ainsi que le porteur d'une telle carte risque de découvrir, lors d'une exploration de routine visant à se faire une idée de ce qu'elle contient vraiment, de grossières failles de sécurité dont il n'aurait jamais osé soupçonner l'existence.

5.1 LA CARTE VITALE : UNE CARTE « SCOT » ?

Après une entrée en scène plutôt laborieuse, la carte Vitale a fini par faire ses preuves, au point d'éveiller l'intérêt d'un certain nombre d'autres pays !

Assez proche de celle de la carte bancaire, sa technologie mérite assurément d'être passée au peigne fin, compte tenu du caractère éminemment personnel des données qu'elle est appelée à gérer.

Nous verrons au prochain chapitre que la carte bancaire B0' est basée sur la technologie BULL CP8, mais qu'elle n'exploite que très imparfaitement (et c'est le moins que l'on puisse dire…) le puissant arsenal sécuritaire dont sa puce est pourtant dotée.

Datant forcément d'avant 1998 (l'année de sa mise en place dans des régions test), la carte Vitale semble avoir été développée sur une base très comparable. Un simple passage au « scanner » (voir page 40) montre, en effet, que la carte Vitale reconnaît sensiblement le même jeu de commandes (avec une « classe ISO » égale à BCh) que sa cousine germaine.

En creusant un peu les choses, il apparaît que la similitude est encore bien plus marquée avec la famille « SCOT », une gamme de cartes BULL CP8 destinée aux applications les plus variées (voir notre ouvrage *PC et cartes à puce*).

Sa technologie correspond ainsi à peu près à ce qui était « l'état de l'Art » en 1991.

Conformément à la bonne vieille stratégie du « rideau de fumée », l'émetteur ne divulgue aucune information technique, en dehors de partenariats avec des industriels « triés sur le volet » (par exemple, les constructeurs des bornes de consultation, des terminaux de professionnels de santé, ou des lecteurs de poche « grand public »).

Qu'à cela ne tienne ! La lecture du manuel (nullement confidentiel) des kits de développement SCOT est fort riche d'enseignements…

Il n'en faut pas davantage pour partir à l'aventure, dans le but avoué (car parfaitement avouable !) d'évaluer le degré de confiance que l'on peut accorder à une carte qui « ne contient aucune donnée confidentielle » (c'est ce que l'on dit aux titulaires), mais dont « les informations administratives qu'elle contient sont codées, par respect de la vie privée » (c'est ce que l'on affirme aux professionnels qui affichent quelques états d'âme).

L'organisation logique des cartes SCOT repose sur des mots de 32 bits (4 octets), mais l'adressage se fait au niveau du quartet (groupe de 4 bits).

Une commande de lecture, transmise par n'importe quel lecteur de cartes à puce asynchrones, sera de la forme `BC B0 P1 P2 LE`, l'adresse à lire étant désignée par les deux octets P1 et P2, le nombre d'octets à lire (maximum 256) étant, quant à lui, indiqué par l'octet LE.

Il faudra donc incrémenter P1P2 de deux unités pour avancer d'un octet (c'est-à-dire de deux quartets), et de huit unités pour avancer d'un mot de 32 bits.

L'espace mémoire d'une carte SCOT est partitionné en plusieurs zones, dont les adresses initiales sont stockées dans autant de pointeurs spécifiques :

```
AD1 : Clefs émetteur
ADS : Jeu secret
AD2 : Codes porteur
ADM : Zone d'accès
ADC : Zone de travail N°2 (dite "confidentielle")
```

```
ADT : Zone de travail N°1
ADL : Zone de lecture
```

Les valeurs de ces pointeurs (et bien d'autres informations utiles) résident dans la zone de lecture qui, comme son nom l'indique, est toujours à lecture libre. Chacun peut donc en prendre connaissance après avoir calculé les adresses où ils résident. Cela nécessite de se procurer la valeur ADMAX (adresse de fin de mémoire + 8h), en envoyant une commande `BC C0 00 00 08` à la carte juste après un « reset ». On trouvera ADMAX dans les troisième (poids fort) et quatrième (poids faible) octets du mot reçu en réponse. Ses deux derniers octets (dits TA et PZT2) renseignent respectivement sur le type d'algorithme cryptographique supporté par la carte, et sur les modalités de protection de la zone de travail N° 2.

Une carte Vitale typique pourra ainsi répondre ADMAX = 2188h, TA = 0Eh, et PZT2 = 08h. En clair, cela signifie qu'elle contient un peu plus de 8 000 quartets (la première adresse accessible étant invariablement 0200h), soit environ 4 000 octets, que la carte supporte le DES (un algorithme pas franchement moderne), et que la zone de travail N° 2 est libre en lecture (mais protégée en écriture), tout comme la zone de travail N° 1.

5.2 DES MYSTÈRES À DISSIPER

Les remarques suivantes s'imposent tout naturellement à un explorateur averti :

- La capacité mémoire est largement supérieure à celle d'une carte bancaire (tout juste 1 024 octets), et pourrait donc potentiellement héberger un volume significatif de données (confidentielles ?).
- L'intégralité des données (cryptées ou non) que contiennent les zones de travail est librement accessible, en lecture, par n'importe quel détenteur, légitime ou non, de la carte.
- Aucun code PIN (code porteur) n'est remis au titulaire, bien que le mécanisme correspondant existe dans les cartes SCOT : d'intrépides expérimentateurs sont d'ailleurs parvenus à bloquer leur carte Vitale en lui envoyant plusieurs commandes `BC 40 00 00 00` (ratification d'un code PIN non présenté, et donc considéré comme erroné), puis à se la faire « débloquer » par les soins de leur caisse (qui a, par conséquent, d'ores et déjà accès aux outils nécessaires) !

Notons qu'en revanche, un code PIN est tout de même indispensable, et c'est la moindre des choses, pour l'utilisation des cartes de professionnel de santé.

Il n'est donc pas interdit d'imaginer qu'une sécurisation supplémentaire par code « porteur » pourrait être instaurée, si le besoin devait s'en faire sentir un jour.

Toujours dans le cas d'ADMAX = 2188h, on trouvera les pointeurs des différentes zones dans le second octet des mots de 32 bits dont voici les adresses, calculées par simple soustraction récursive de la valeur 08h :

```
AD1 : 2168h
ADS : 2158h
AD2 : 2150h
ADM : 2148h
ADC : 2140h
ADT : 2138h
ADL : 2130h
```

En prime, on trouvera le numéro de série de la carte (sur 25 bits) à l'adresse 2178h, un certain nombre de paramètres techniques aux adresses 2180h et 2170h, et le « type d'application » (sur 14 bits) en 2160h.

Nous avons ainsi fait le tour complet de ce qu'il est convenu d'appeler la « zone de fabrication », qui suit immédiatement la « zone de lecture ».

En présence d'une carte SCOT « inconnue », notre programme INFOSCOT.BAS (naturellement développé en ZCBasic !) se fera un plaisir de déterminer automatiquement les valeurs de tous ces pointeurs.

Également fournie dans le dossier « VITALE » du cédérom, sa version exécutable INFOSCOT.EXE fonctionnera directement sur n'importe quel lecteur PC/SC (`ComPort=101`).

Elle pourrait éventuellement être recompilée pour le lecteur « CyberMouse » fourni dans les kits BasicCard (`ComPort=1` s'il est branché sur COM1:, ou `ComPort=2` s'il est relié à COM2:).

Muni de tous ces précieux indices, il n'est guère compliqué de procéder à la lecture exhaustive de la carte, exception faite des clefs secrètes (naturellement protégées en lecture !) et de la « zone d'accès » (qui ne sert guère qu'à tenir la comptabilité des présentations, réussies ou non, des clefs et codes PIN).

5.3 UNE NÉCESSAIRE SURVEILLANCE

Notre programme COPYSCOT.BAS a été conçu pour faciliter différentes opérations autour des cartes SCOT en général, et donc pas spécialement des cartes Vitale.

Il n'est pas très rapide, car il ne lit (volontairement) qu'un seul mot de 32 bits par commande `BC B0`, en l'enregistrant aussitôt dans un fichier baptisé CLOVIS.LOG.

À ce stade de notre expérimentation, on quittera le programme (en pressant ESCape) lorsque la lecture sera terminée, et on mettra éventuellement « en lieu sûr » une copie de CLOVIS.LOG.

Un autre programme (CHKSCOT.BAS) pourra, en effet, servir ultérieurement à mettre en évidence d'éventuelles modifications du contenu de la carte ; cela, après une « mise à jour » dans une borne à ce destinée, par exemple, car dans l'état actuel des choses, il ne semble pas qu'un terminal de professionnel de santé soit habilité à modifier quoi que ce soit (mais cela pourrait venir).

La carte étant de nouveau insérée dans le lecteur PC/SC, il suffit de lancer CHKSCOT.EXE (en présence de CLOVIS.LOG) pour assister à une comparaison mot à mot entre la carte et le fichier.

En cas de discordance, le programme s'arrêtera pour laisser le temps de la noter, mais il reprendra son cours normal dès que l'on pressera la touche de retour à la ligne :

```
2C9A0585      2C9A0585
265985A0      265985A0
04007FFF      04007FFF
37FFFFFF      37FFFFFF
00000000      00000000
010212FF      01021200
```

Comme l'affichage se fait en hexadécimal, et à un rythme pas trop rapide, il y a là une bonne occasion de se faire une idée de ce que contient vraiment la carte Vitale.

Il sautera immédiatement aux yeux que seule une toute petite partie de la mémoire disponible est réellement utilisée !

Cela confirme bien que, dans l'état actuel des choses, la carte ne contient certainement pas de données purement médicales, telles qu'un équivalent de l'inénarrable « carnet de santé ».

Il semblerait plutôt qu'elle rassemble un certain nombre de « fiches », avec pour chacune d'elles un numéro de sécurité sociale et une date de mise à jour (format « année, mois, jour »).

Une première fiche identifierait le titulaire de la carte (chef de famille), les suivantes correspondant sans doute chacune à l'un de ses ayants droit.

D'après ce que parviennent à lire les bornes de consultation et les lecteurs de poche, il est clair que chaque fiche contient aussi des informations plus « confidentielles » : état civil (nom, prénoms,

date de naissance), validité des droits, caisse de rattachement, etc. Rien de plus, en somme, que ce qui était imprimé sur les cartes « papier » !

Compte tenu du faible nombre d'octets de chaque fiche, il se confirme qu'un semblant de codage doit être appliqué, opérant au passage une certaine forme de compression.

Il est toutefois intéressant de remarquer que tous les lecteurs savent procéder au décodage correspondant, vraisemblablement pas bien compliqué, même s'ils fonctionnent indépendamment de la « carte de professionnel de santé » (CPS).

Le rôle de celle-ci se bornerait-il donc à sécuriser seulement l'émission et la télétransmission des « feuilles de soins électroniques », ou bien servirait-elle tout de même de clef d'accès à la partie la plus confidentielle des données personnelles, présentes ou à venir ?

5.4 DES COPIES DE SAUVEGARDE ?

La BasicCard a cela d'extraordinaire, que quelques lignes de Basic suffisent pour lui faire émuler, dans certaines limites, le jeu de commandes de telle ou telle carte à puce existante, avec souvent un surcroît de souplesse.

Par exemple, EMUSCOT.BAS est un court programme qui, chargé dans une BasicCard à protocole « T = 0 » (ZC 5.4 ou ZC 5.5), la dote des commandes nécessaires pour être lisible dans les mêmes conditions qu'une carte SCOT.

La commande d'écriture, cependant, reste volontairement très rudimentaire, et ne pourra donc guère servir qu'à initialiser la mémoire, par mots entiers de 32 bits.

Cela, à partir de données recueillies, mettons, dans une carte à lecture libre, ou bien choisies en vue de provoquer, sur un lecteur, tel ou tel phénomène potentiellement intéressant à analyser.

La loi autorisant expressément les « copies de sauvegarde » dans une grande variété de situations, il serait dommage de se priver du petit plaisir de voir ce que cela donne avec une carte Vitale !

C'est là que la seconde partie de COPYSCOT.EXE va entrer en scène : au lieu de quitter le programme après avoir lu l'original, on remplacera celui-ci par une ZC 5.4 ou 5.5, dans laquelle on aura préalablement chargé le fichier EMUSCOT.IMG (à l'aide de BCLOAD.EXE).

La structure mémoire du « modèle » sera scrupuleusement respectée, les différents pointeurs demeurant inchangés.

Par rapport à l'original, la « copie de sauvegarde » ainsi réalisée perd bien sûr toutes les données qui étaient éventuellement protégées en lecture, au minimum les différents jeux de clefs secrètes.

De plus, aucune commande à vocation sécuritaire (et il y en a beaucoup dans les cartes SCOT !) n'est émulée.

Pas question, par conséquent, de faire mettre à jour une telle « photocopie » d'une carte Vitale dans une borne publique, opération qui serait d'ailleurs assimilable à une falsification.

Par contre, le fait que le premier lecteur venu arrive à lire indifféremment l'original ou la copie démontre, à l'évidence, que le décodage des principales données personnelles ne fait appel ni aux clefs secrètes confiées à la carte, ni aux algorithmes cryptographiques dont elle dispose. Un exemple de plus, donc, d'une application « sensible » qui ne profite que bien partiellement des ressources sécuritaires des cartes qu'elle utilise !

5.5 ET AVEC UN LECTEUR DE POCHE…

Un tel constat donne forcément envie de pousser plus loin les investigations, en étudiant tout bonnement comment les lecteurs de poche, librement vendus au grand public, communiquent avec la carte Vitale.

Il est d'ailleurs intéressant de faire la distinction entre ceux-ci, et les « consulteurs » conçus pour les professionnels prodiguant des soins à domicile.

Les lecteurs les plus simples (XL 2000 de Xiring ou Lexibook, par exemple) opèrent aussi bien sur les cartes bancaires (consultation de l'historique des transactions) ou les télécartes (lecture du solde d'unités), que sur les cartes Vitale (numéro d'immatriculation, centre d'affiliation…).

Sous une apparence très similaire, les « consulteurs » affichent aussi des informations plus « indiscrètes », telles que le détail des droits attachés à la carte.

En gros, un consulteur donne accès aux données qui étaient imprimées sur la carte « papier », tandis qu'un lecteur affiche plutôt l'équivalent de ce qui apparaissait sur un « extrait de carte », expurgé de toute information confidentielle.

Notons pourtant que les dernières générations de lecteurs de poche (XL 2500) en font presque autant !

Déterminer, à l'aide d'un « espion de cartes asynchrones », où le lecteur va lire les données qu'il doit afficher, est l'affaire d'un instant.

Une autre méthode consiste à programmer la BasicCard pour qu'elle enregistre, dans sa propre mémoire, toutes les commandes qu'elle reçoit du lecteur : il suffit, pour ce faire, d'y charger SPYSCOT.IMG au lieu d'EMUSCOT.IMG.

L'utilitaire SPYUTIL.EXE servira à activer ou désactiver ce mécanisme, à récupérer l'enregistrement dans un fichier (CARD.LOG) sur le disque dur du PC, et enfin à libérer la mémoire de la carte après usage.

On découvrira ainsi que bien peu de commandes de lecture suffisent, à la plupart des lecteurs, pour accéder à tout ce dont ils ont besoin.

Cela pourrait être :

```
BC B0 21 30 28
BC B0 20 C8 30
BC B0 02 98 04
BC B0 03 50 1C
BC B0 02 F0 30
```

Dès lors, il suffit d'examiner attentivement les données (très personnelles !) retournées par la carte, pour s'apercevoir que presque tout est finalement enregistré « en clair ».

Simplement, les lettres (uniquement des majuscules) sont codées sur seulement 5 bits, selon le **tableau 5.1**.

Tableau 5.1. Le codage des lettres.

		P	10000
A	00001	Q	10001
B	00010	R	10010
C	00011	S	10011
D	00100	T	10100
E	00101	U	10101
F	00110	V	10110
G	00111	W	10111
H	01000	X	11000
I	01001	Y	11001
J	01010	Z	11010
K	01011		
L	01100		
M	01101		
N	01110		
O	01111		

Les chiffres, pour leur part, sont très classiquement codés en BCD, sur 4 bits, selon le **tableau 5.2**.

Tableau 5.2.
Le codage « BCD » des chiffres.

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

La seule véritable complication, c'est que chaque mot de 32 bits commence par 2 bits à 0, ce qui décale tout à chaque fois.

À ce stade, on pourra être tenté de pousser plus loin la plaisanterie : modifier le contenu de la carte, et examiner l'effet produit sur ce qu'affiche le lecteur de poche.

Cela uniquement dans la copie, bien sûr, car l'écriture dans une « vraie » carte est heureusement subordonnée à la présentation de la clef *ad hoc*.

Supposons que le contenu du mot d'adresse 0318h soit `01 A4 92 35`, et que le prénom du porteur de la carte soit « PATRICK ».

Remplaçons l'octet A4h (`10100100`) par 9Ch (`10011100`), ce qui se fait très simplement en envoyant une commande `BC D0 03 18 04 01 9C 92 35` (notre carte attendant, rappelons-le, des mots entiers de 4 octets, à des adresses finissant par 0 ou 8).

Insérons par exemple la carte dans un lecteur de poche XL 2500, et il s'affichera maintenant « PASRICK » !

Rien de plus normal, quand on sait que le T est codé `10100`, et le S `10011`.

Mais cette expérience a au moins le mérite de montrer que le lecteur n'y voit que du feu : n'y aurait-il donc aucune « clef de contrôle » à ce niveau, ou bien le lecteur ne se donne-t-il même pas la peine de la vérifier ?

Espérons tout de même que les données matérialisant les droits du porteur soient traitées avec un peu moins de désinvolture…

5.6 DES EXPÉRIENCES D'AUTHENTIFICATION

Même s'il ne sert sûrement pas autant qu'il le pourrait, un mécanisme d'authentification assez performant réside bel et bien dans la carte Vitale.

Il est extrêmement facile de le faire jouer, tout comme nous l'avons fait au chapitre 4 pour les cartes SIM des téléphones portables : tout le nécessaire se trouve dans le répertoire « AUTH » du dossier « VITALE » de notre cédérom.

Notre programme AUTHVIT.BAS permet de constater qu'il fonctionne d'une façon assez sensiblement différente, établissant en quelque sorte un « certificat » à partir du contenu d'une adresse carte librement choisie.

Cela, en lui combinant une valeur aléatoire de 48 bits transmise par le lecteur (ici, la chaîne « AUTHEN »), et naturellement une clef secrète qui n'est jamais directement lisible.

À noter que si l'adresse carte à « certifier » résidait dans une zone protégée en lecture par un code confidentiel « porteur », il faudrait tout d'abord présenter et ratifier celui-ci.

Authentifiant à la fois la carte et son porteur, cela garantirait un fort bon niveau de sécurité, mais… enfreindrait gravement la « loi du moindre effort » !

Bien entendu, chacun pourra faire varier à sa guise le mot aléatoire et l'adresse carte, afin de vérifier le caractère théoriquement imprévisible du résultat obtenu.

On pourra aussi s'assurer (même si c'est absolument évident !) qu'une carte copiée comme nous l'avons vu est parfaitement incapable de simuler ce processus.

Rien à voir, donc, avec un « clone » potentiellement illégal : tout au plus l'équivalent de la simple photocopie d'une carte « papier », sur laquelle on peut crayonner à loisir.

Vraiment pas de quoi fouetter un chat, en somme !



CHAPITRE PAGE

6 AUTOUR DES CARTES DE PAIEMENT

La « monétique » a toujours été l'une des applications de prédilection des cartes à puce, et ce n'est certes pas fini !

Parallèlement aux cartes bancaires, il faut compter avec une multitude de cartes permettant de régler commodément de petits montants dans les parcmètres, publiphones, distributeurs automatiques, automates de péage, etc.

Cela, en attendant qu'un « porte-monnaie électronique » digne de ce nom vienne fédérer, dans des conditions enfin acceptables, toutes ces applications résolument disparates.

Et n'oublions pas que les opérateurs de téléphonie mobile rêvent de concurrencer les banques, en offrant des possibilités de « micropaiements » à partir de leurs formules prépayées.

6.1 L'ÉVOLUTION DES CARTES BANCAIRES

Confrontée aux problèmes de sécurité que l'on sait, la carte bancaire évolue lentement mais sûrement.

En attendant la généralisation de la technologie « EMV », le masque B0' a déjà subi nombre de « replâtrages », qu'il est intéressant de mettre en évidence par des manipulations finalement fort simples.

Et n'en déplaise aux mauvaises langues, la carte bancaire s'est très sensiblement perfectionnée au fil des années, moins vite toutefois que la technicité des attaques dont elle est la cible.

Il faut dire qu'avec une périodicité de renouvellement de deux ans (il n'y a pas de petites économies…), l'inertie du système demeure considérable.

La première révolution technologique a été le passage du masque B0 (à technologie « EPROM OTP ») au masque B0' (à technologie EEPROM, autrement dit réinscriptible).

Endormie sur ses lauriers pendant quelques années, la puce B0' a renoué, vers 1995, avec une évolution qui n'a plus guère cessé depuis. Il est facile de s'en rendre compte en allant lire les valeurs d'une poignée de « pointeurs » dans les cartes que l'on a peut-être eu la (bonne ?) idée de ne pas détruire au fil de leurs renouvellements successifs.

Il faut savoir que la mémoire d'une carte B0' se compose, en tout et pour tout, de 256 mots de 32 bits, soit un total de 1 024 octets.

Leur adressage se fait curieusement par quartets (façon « Bull CP8 ») plutôt que par octets, et en aucune façon par sélection de fichiers (façon GSM ou EMV).

Notons au passage qu'en raison de la présence de bits de contrôle, chaque mot compte généralement moins de 32 bits de « charge utile ».

La première adresse définie est 0200h, tandis que la première adresse physiquement non existante est 0A00h.

La différence entre ces deux valeurs hexadécimales est 0800h, ce qui correspond bien à 2 048 quartets ou 1 024 octets.

L'adresse 09C0h joue un rôle clef, dans le sens où elle marque le début de la seule zone fixe de l'espace mémoire de la carte.

Cette « zone de fabrication », à lecture libre, contient un certain nombre de paramètres techniques, à commencer par des « pointeurs » définissant les adresses de début de toutes les autres partitions.

6.2 PAR OÙ COMMENCER L'EXPLORATION ?

N'importe quel lecteur de cartes à puce supportant le protocole asynchrone « T = 0 » permet de prendre connaissance du contenu de la zone de fabrication, au moyen de simples commandes ISO 7816.

On fera ainsi `BC B0 09 C0 04` pour lire le premier mot de 32 bits (soit 4 octets) de cette zone, `BC B0 09 C8 04` pour lire le suivant, ou `BC B0 09 F8 04` pour lire le dernier.

Mais on pourrait tout aussi bien faire `BC B0 09 C0 20` pour lire d'un seul coup les 32 octets de la zone.

Les heureux possesseurs d'un lecteur PC/SC, convenablement installé sur un système Windows 95 ou supérieur, pourront se servir pour cela de SCARD.EXE, ou mieux de notre programme ZF.EXE, capable de décoder, en langage clair, l'essentiel de ce que contient la zone de fabrication (voir dossier « CB » du cédérom accompagnant cet ouvrage).

Développé en ZCBasic version 4, son code source ZF.BAS est compatible avec le kit logiciel de la BasicCard, également offert sur notre cédérom.

Soumise à ce logiciel, une carte bancaire récente (émise en juin 2002) donnera par exemple les résultats suivants :

```
Code Application :                    3FE5
ZF (Zone de fabrication) :            9C0
ADL (Zone de lecture) :               7F0
ADT (Zone de transaction) :           2F8
ADM (Zone d'état) :                   2D8
```

```
ADC (Zone confidentielle) :          2F8
AD1 (Zone des clefs) :               210
ADS (Clef de transaction) :          230
AD2 (Zone des codes) :               2C0
Numéro de Fabricant :                1
```

Avec une carte émise quatre ans plus tôt par la même banque, on pourrait obtenir :

```
Code Application :                   3FE5
ZF (Zone de fabrication) :           9C0
ADL (Zone de lecture) :              8E0
ADT (Zone de transaction) :          2B0
ADM (Zone d'état) :                  290
ADC (Zone confidentielle) :          2B0
AD1 (Zone des clefs) :               210
ADS (Clef de transaction) :          230
AD2 (Zone des codes) :               278
Numéro de Fabricant :                2
```

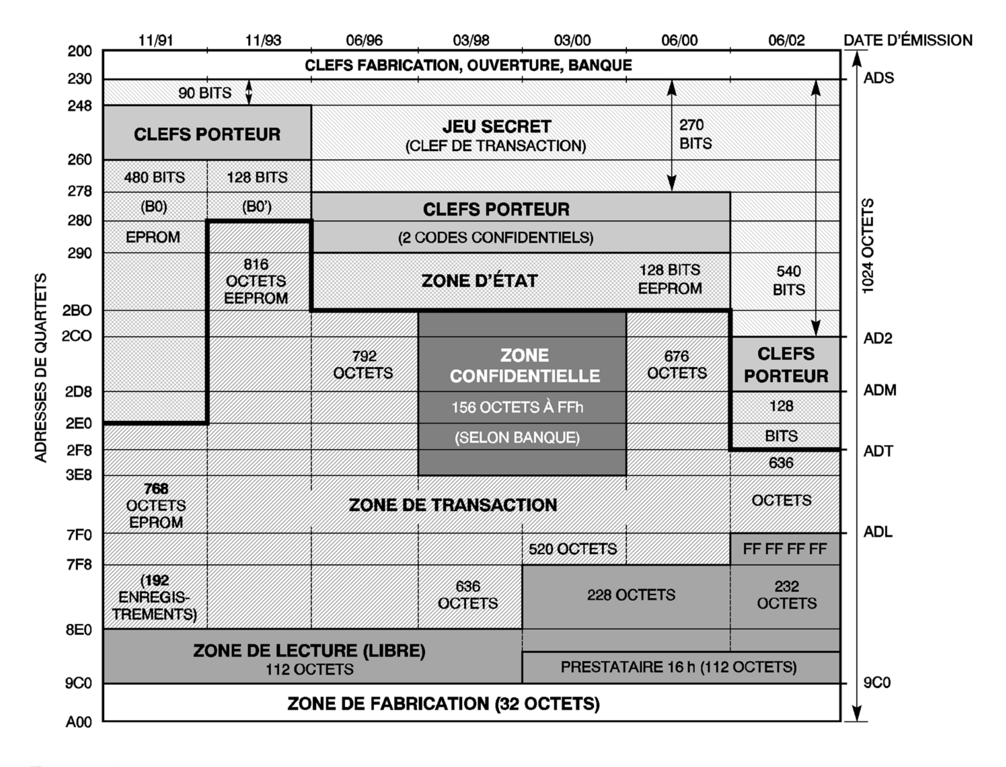


Figure 6.1.
La carte bancaire à travers les âges…

La **figure 6.1** résume, pour sa part, les résultats d'investigations menées de cette façon sur sept cartes émises, en près de dix ans, par deux banques différentes, ainsi que les déductions que l'on peut raisonnablement se risquer à en tirer.

Elle met bien en évidence la grande « élasticité » de l'organisation de la mémoire, et la façon dont les émetteurs en ont très largement tiré parti.

On remarquera, tout d'abord, que le masque B0 se caractérisait par une « zone d'état » particulièrement vaste (480 bits), car non effaçable.

Un bit y étant irréversiblement « grillé » lors de chaque présentation du code confidentiel à la puce, les cartes de cette époque rendaient souvent l'âme avant la fin de leurs deux ans de validité.

De même, à raison d'un enregistrement de 32 bits par opération, une « zone de transaction » de 768 octets ne permettait guère plus de deux paiements « puce » par semaine (mais à l'époque, les facturettes imprimées au « fer à repasser » étaient encore monnaie courante).

Ce n'est guère que lors du passage au masque B0' que les banques ont, enfin, reconnu publiquement l'existence de phénomènes de « saturation » des puces, et par là même la présence d'un historique des opérations dans la carte.

La technologie EEPROM permettant de « recycler » aussi souvent que nécessaire la zone d'état, celle-ci a alors pu être réduite à 128 bits, au grand bénéfice de la zone de transaction, d'une capacité record de 816 octets (80 % de la mémoire disponible) avant recyclage.

Cela, jusqu'à ce que l'on s'aperçoive qu'une clef cryptographique de 90 bits ne présentait plus des garanties de sécurité suffisantes, d'où une extension à 270 bits de ce « jeu secret ».

En contrepartie, la zone de transaction est tombée à la capacité encore confortable de 792 octets, sauf chez les banques se réservant une « zone confidentielle » de 156 octets.

Souvent complètement vierge (à FFh), cette zone pourrait bien avoir un rapport avec certains projets, parfois mort-nés, de « porte-monnaie électronique ».

L'étape suivante concerne la « zone de lecture », dans laquelle résident des informations équivalentes à ce qui est embossé dans le plastique (numéro, identité du porteur, date de validité), ainsi que la fameuse « valeur d'authentification » de 320 bits (dite « prestataire 03h »).

L'apparition, dans cette zone à lecture libre, d'un bloc de 112 octets (+ 4 octets d'en-tête : `2E 16 70 3A`), baptisé « prestataire 16h », fait penser à une seconde valeur d'authentification, d'une longueur tournant cette fois autour de 784 bits utiles.

Cela serait assez cohérent avec l'introduction supposée, à compter de novembre 1999, d'une clef RSA de 768 bits.

Parallèlement, le « jeu secret » des dernières cartes émises semble bien être passé à 540 bits, tout cela ne laissant plus guère que 636 octets pour la zone de transaction.

C'est encore tout juste suffisant pour permettre, comme promis, l'affichage des 140 dernières opérations sur les lecteurs de poche disponibles dans le commerce, mais il ne doit plus rester beaucoup de marge de manœuvre d'ici à l'EMV (à moins de cannibaliser carrément cet historique ?)

Mais la zone de lecture héberge encore bien d'autres informations intéressantes, dont le décodage n'est cependant pas facile, car comme dans la carte Vitale, leurs bits ne sont pas jointifs.

Toujours développé en ZCBasic, notre programme ADL.EXE permet à tout lecteur PC/SC de créer un fichier ADL.CAR, 100 % compatible avec le logiciel de décodage DECADL publié dans notre ouvrage *PC et cartes à puce*, ou bien avec son équivalent compilé sous ZCBasic : ZCDECADL.EXE.

Dans un cas comme dans l'autre, on obtiendra un résultat ressemblant à ceci :

```
DECADL  (c)1997 Patrick GUEULLE     ANALYSE DU FICHIER ADL.CAR

CARTE N°:         4  9  7  0  0  0  0  0  0  0  0  0  0  0  0  0
Code service:     1  0  1
Début validité:   9  3  1  1
Code langue:      2  5  0     (250 = français)
Fin de validité:  9  5  1  1
Code devise:      2  5  0     (250 = francs français)
Exposant:         3           (3 = centièmes, 5 = unités)
Porteur:          MR PATRICK

Mot d'option:     1  1  1  1  1  1  0  1  0  0  0
ADL = 08E0   ADT = 0280   ADC = 0280   ADM = 0260   AD2 = 0248
ADS = 0230
Ep =  0       (Zone de transaction protégée en écriture)
Lp =  0       (Zone de transaction protégée en lecture)
AD1 = 0210
N° de fabricant:  1  (CP8 Oberthur)
N° de série:
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
```

```
Mot des verrous:  1  0  0  1  1    (10011 = carte en phase
d'utilisation)
```

6.3 L'HISTORIQUE DES TRANSACTIONS

Occupant entre 50 et 80 % de la capacité mémoire de la puce, la zone de transaction ne peut être lue qu'après présentation et ratification d'un code confidentiel valide.

Il est donc bien clair que le porteur de la carte est parfaitement fondé à prendre connaissance de son contenu, soit avec un lecteur du commerce (Xiring, Lexibook, etc.), soit, (et pourquoi pas ?) au moyen d'un logiciel pour PC.

Dès 1995, l'auteur révélait dans son ouvrage *PC et cartes à puce* comment procéder à l'opération avec un lecteur à construire soi-même.

Les choses étant naturellement plus confortables avec un lecteur PC/SC, une application ZCBasic (ADT.EXE) a été développée depuis pour décharger, sur disque, l'intégralité de la zone de transaction de toute carte bancaire suffisamment récente.

Cette restriction est liée au fait que, pour des raisons matérielles, certaines puces un peu anciennes ne sont pas correctement supportées par les lecteurs les plus récents, qui ne peuvent ainsi accéder aux zones protégées par le code confidentiel.

Le fichier ADT.LOG créé par ce programme est un véritable « relevé de compte » en langage clair, qui peut être consulté ou imprimé avec n'importe quel éditeur de texte :

```
---------- Mois 0204 ----------
Le 3 :        157
Le 16 :       110
Le 22 :       150
---------- Mois 0205 ----------
Le 13 :       143
Le 29 :       363
---------- Mois 0206 ----------
Le 4 :        500
Le 4 :        518
Le 10 :       58
Le 13 :       500
Le 13 :       518
Le 14 :       104
Le 24 :       533
Le 24 :       500
Le 24 :       107
```

C'est à ce stade que l'on aura la croustillante surprise de découvrir que bien des cartes émises courant 2002 enregistrent toujours en francs les paiements effectués en euros !

On remarquera aussi qu'elles enregistrent 200 F, 500 F, ou même 800 F tout rond pour tout achat de carburant effectué dans une station-service automatisée.

L'explication est simple : comme le client récupère sa carte avant d'aller se servir, on y enregistre un montant arbitraire, grassement arrondi à hauteur de l'autorisation obtenue.

L'histoire ne dit pas de combien serait débité le compte bancaire en cas d'incident technique entraînant la perte du montant réel…

Pour un maximum de confort, le répertoire « HISTORIQUE » du dossier « CB » de notre cédérom contient une application Windows en bonne et due forme, capable d'afficher et imprimer l'intégralité de l'historique d'une carte bancaire B0', insérée dans un lecteur PC/SC correctement installé.

Développé en Delphi 3 (code source fourni dans le sous-répertoire « DELPHI »), ce logiciel sous-traite l'essentiel de son travail à l'application ZCBasic, appelée par un mécanisme (CreateProcess) qu'il n'est pas inintéressant d'examiner.

Le même subterfuge pourrait, en effet, être employé dans bien d'autres circonstances, pour développer en PC/SC sous ZCBasic, sans pour autant renoncer à l'interface graphique Windows.

Reste que pour l'utilisateur final, l'installation est extrêmement simple : il suffit de double cliquer sur SETUP.EXE, et un assistant créé avec Installshield Express se charge de tout !

Mais pour les inconditionnels du mode « console », un fichier batch (CB.BAT) permet d'enchaîner la panoplie complète des exécutables compilés avec le kit BasicCard 4 :

```
SET ZCPORT=101
ADL.exe
ZCdecADL.exe
ADT.exe
Type ADT.log|More>CON
```

En présence d'un lecteur PC/SC, on obtiendra d'abord l'affichage en clair de tout le contenu de la zone de lecture, zone de fabrication comprise, de la carte bancaire que l'on voudra bien y introduire.

Après présentation du code confidentiel, on visualisera aussi le contenu de la zone de transaction (en Francs ou en centimes, selon que le paramètre « exposant » vaut 5 ou 3).



L'opération laissera, sur le disque, les habituels fichiers ADL.CAR et ADT.LOG, qu'il n'est évidemment pas interdit d'exploiter par d'autres moyens.

6.4 DES EXPÉRIENCES D'AUTHENTIFICATION

Appartenant, en quelque sorte, à la même grande famille, les cartes bancaires supportent un mécanisme d'authentification rappelant fortement celui des cartes Vitale.

Simplement, notre programme AUTHCB.BAS prend en compte le plan mémoire particulier du masque B0', et lui fait ainsi certifier le contenu d'une adresse différente (09E0h au lieu de 2130h).

L'expérience est cependant plus édifiante que dans le cas de la carte Vitale, dans la mesure où l'explorateur avisé a généralement pris le soin de conserver toutes ses cartes bancaires, au fil de leurs renouvellements successifs.

Bien qu'il soit parfaitement impossible de lire leur « jeu secret », il est facile de déterminer si celui-ci change ou non lors de chaque renouvellement de carte (soit, en principe, tous les deux ans) : il suffit de vérifier si notre programme donne ou non un résultat différent !

Et aussi « gros » que cela puisse paraître, chacun pourra aisément se convaincre que, jusqu'à la récente augmentation de la taille de la clef, les banques ne semblent pas avoir déployé beaucoup de zèle en la matière…

Curieusement, cette paresse procure une belle occasion au « bricoleur » qui envisagerait de réaliser son propre système d'authentification, à partir de « vieilles » cartes bancaires ! Point n'est besoin de déployer de bien gros moyens pour ce faire, comme en témoigne la simplicité du code source ZCBasic CB1.BAS.

Compilé en CB1.EXE, ce très court programme commence par demander à son utilisateur de taper un texte quelconque (maximum 256 caractères).

Dans celui-ci, le programme va prélever des chaînes réputées « aléatoires » de 6 octets, qu'il soumettra à l'algorithme cryptographique de la carte, exécuté sur l'adresse 09E0h (mais chacun est libre de changer cette valeur « par défaut »).

Après chaque exécution, le programme enregistre cet « aléa » et la réponse correspondante (8 octets, soit 64 bits), dans un fichier baptisé « KEY.KEY ».

Par la suite, on pourra authentifier la même carte (ou bien, et c'est tout l'intérêt de la « négligence », une carte distincte munie du même jeu secret), en lui soumettant un aléa extrait du fichier KEY.KEY, et en vérifiant que le résultat obtenu coïncide avec celui qui lui est attaché dans ce même fichier.

C'est très exactement ce que fait notre programme CB2.BAS, ou sa version compilée CB2.EXE !

En toute rigueur, la sûreté du système suppose que l'on n'utilise chaque aléa (ou « numéro de clef ») qu'une seule et unique fois.

Remarquons tout de même qu'un texte aléatoire de 256 octets peut être scindé en 42 chaînes de 6 octets, ce qui suffit amplement pour générer un couple aléa-résultat chaque jour, pendant un bon mois.

On peut donc imaginer que, chaque mois, l'on doive « réhabiliter » la carte en lui faisant produire une trentaine de couples aléa-résultat, dont on utilisera un nouveau chaque jour pour accéder à un local, ou pour démarrer un poste de travail informatique, au lieu d'afficher simplement « Carte reconnue ! ».

Cela, en se servant tout simplement de sa carte bancaire habituelle, périmée ou en cours de validité, ce qui ne lui fera aucun mal.

Et pour aller encore plus loin dans la voie de la sécurité, pourquoi ne pas imaginer d'effacer du fichier KEY.KEY tout couple ayant servi une seule fois, ou encore de produire les couples à partir d'une adresse nécessitant la présentation du code confidentiel de la carte ?

On authentifierait ainsi non seulement la carte, mais aussi son porteur.

Enfin, n'oublions pas l'approche qui consisterait, nous y voila, à remplacer le fichier KEY.KEY par une seconde carte munie du même jeu secret, insérée dans son propre lecteur !

On viendrait alors comparer sa réponse à celle de la carte présentée par l'utilisateur ; une variante, en somme, du « module de sécurité » dont sont équipées les applications les plus « pointues », offerte « sur un plateau » par notre banque habituelle…

6.5 VERS LES CARTES « EMV »

Dans un communiqué diffusé en mai 2002, les organisateurs du salon CARTES citaient une déclaration d'un administrateur du GIE Cartes Bancaires : « *Les banques françaises vont investir environ 1 milliard d'euros sur 7 à 8 ans pour assurer le passage des cartes et des terminaux à la norme EMV* ».

Cela, alors que les cartes et les terminaux supportant cette nouvelle technologie existaient d'ores et déjà, puisqu'une transaction EMV avait été réussie, dès le 7 février 2002, chez un commerçant français.

Avec une carte bancaire… anglaise, cependant, 30 millions de cartes EMV circulant déjà, à cette époque, outre-Manche.

En fait, le délai annoncé avoisine le temps jugé nécessaire pour achever le programme de migration au niveau mondial, et en particulier aux États-Unis (où pas moins de 800 millions de cartes à pistes magnétiques sont concernées).

En France, il faut s'attendre à rencontrer des cartes à technologie mixte B0'/EMV pendant une période transitoire qui pourrait bien « traîner » jusqu'en 2006, voire au delà, et c'est tant mieux pour nos expérimentations…

Fonctionnellement plus riches, plus sophistiquées, plus sécurisées (dit-on), les nouvelles cartes sont surtout plus coûteuses (3,5 à 4 € contre 1,50 € pour les actuelles cartes B0').

Elles font, soit dit en passant, l'objet d'une normalisation très ouverte, et de nature résolument publique, ce qui est assurément une bonne chose.

Certains experts craignent toutefois que les mesures de sécurité (facultatives) qu'elle définit ne soient (pour changer ?) mises en œuvre que d'une façon maladroite, ou même pas du tout !

Cela se traduirait, naturellement, par un résultat diamétralement opposé au but poursuivi (le renforcement de la sécurité).

Parallèlement, on commence à intégrer le porte-monnaie électronique « Monéo » dans les cartes bancaires, ce qui fait couler beaucoup d'encre !

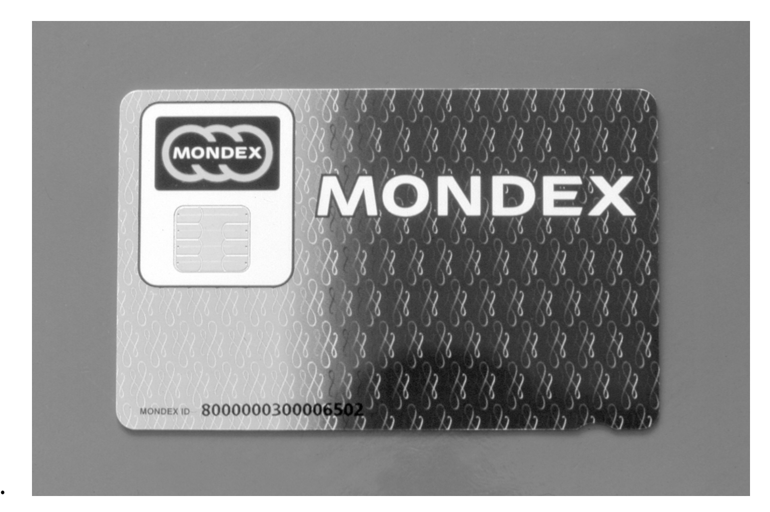
6.6 ET VOICI « MONÉO » !

Par définition, un porte-monnaie électronique (PME) est censé se substituer à la menue monnaie pour effectuer commodément, par carte, des paiements de faibles montants.

En toute logique, il devrait donc être « sans abonnement », anonyme, et rechargeable aussi bien par débit d'un autre PME (parents vers enfants, par exemple) que d'un compte bancaire.

Un tel produit existe (il s'appelle « Mondex »), et a amplement fait ses preuves, notamment en Grande-Bretagne.

Il n'avait, hélas, que fort peu de chances de s'imposer au pays de Colbert…



MONDEX : peut-être le porte-monnaie électronique idéal…

À l'heure où l'auteur écrit ces lignes, tout laisse à penser qu'un PME franco-français payant, centralisateur, et potentiellement indiscret, ne fera pas vraiment l'unanimité.

Faudra-t-il donc l'incorporer « de gré ou de force » dans les cartes bancaires, lors de leur renouvellement (que l'on peut cependant toujours refuser…) ?

Mais le plus « croustillant », c'est qu'avant même d'avoir seulement touché une carte Monéo, on peut déjà commencer un semblant d'exploration !

Cela, en mettant à contribution des lecteurs de poche, à commencer par celui qui accompagne les kits BasicCard du commerce.



Le lecteur de poche (Pocket Reader) des kits BasicCard.

D'une façon générale, ce « Pocket Reader » reste trop souvent inemployé, et c'est fort dommage !

Il faut dire que l'application programmée dans la carte de démonstration qui l'accompagne ne met pas vraiment l'eau à la bouche.

Elle a, tout de même, l'intérêt de montrer les différentes variantes de format d'affichage que supporte le lecteur, sur son écran LCD à une seule ligne de dix caractères.

Les quatre premières positions peuvent afficher à peu près n'importe quel symbole alphanumérique, tandis que les six dernières sont limitées à des caractères numériques (hexadécimaux, c'est-à-dire y compris les lettres A à F) et aux signes + et –.

Un point décimal (faisant office de « virgule ») peut également être inséré dans les valeurs numériques.

Deux points essentiels doivent être parfaitement compris avant de tenter d'utiliser le lecteur :

- Il ne supporte que les cartes à protocole « T = 1 » (BasicCard « Compact » ZC 1.X et « Enhanced » ZC 3.X, ou « Professional » ZC 4.X / ZC 5.X en mode T = 1, etc.).
- Tout échange entre lecteur et carte se fait au moyen de la seule et unique commande dont la classe est C8h, et le code opératoire 00h.

En pratique, dès qu'une carte asynchrone (et pas forcément une BasicCard) est introduite dans le lecteur, celui-ci lui envoie une commande T = 1 de la forme `C8 00 00 00 Lc DATA Le`.

Si la carte ne comprend pas cette commande, elle renvoie un code d'erreur SW1SW2, que le lecteur affiche *in extenso* (par exemple « `6E00 Error` » ou « `6D00 Error` »). Cela pourra déjà servir à l'occasion !

Si aucun échange n'est possible (cas, notamment, d'une carte synchrone), il s'affiche simplement « `READ Error` ».

Si le lecteur a besoin d'un petit délai pour opérer (car il utilise une fréquence CLK réduite à 1 MHz), il peut aussi afficher « `WAIT` ».

En protocole T = 1, il est courant qu'une même commande soit à la fois « entrante » et « sortante », ce qui est un peu déroutant lorsque l'on n'a jamais opéré qu'en T = 0 (cartes bancaires, cartes SIM, carte Vitale, etc.).

Cela n'est pourtant pas très compliqué, et finalement plutôt commode : l'octet Lc indique la longueur du bloc de données (DATA) que le lecteur transmet à la carte, tandis que Le indique

le nombre d'octets que le lecteur demande à la carte de lui retourner (rien n'interdisant bien sûr à Le d'être égal à Lc, qui vaudra ici 10h, soit 16 en décimal).

Les données que le lecteur envoie ainsi spontanément à la carte décrivent tout simplement ce qu'il est capable de faire.

Comme il n'existe pour l'instant qu'un seul modèle, on peut carrément les ignorer, à l'exception du tout premier octet, qui précise combien d'opérations d'affichage ont été effectuées depuis que la carte est présente dans le lecteur.

Pour faire s'afficher une ligne de caractères, la carte doit simplement renvoyer le bloc d'octets « DATA » qu'elle a reçu, en y apportant les modifications suivantes :

- 1er octet : doit être répété à l'identique.
- 2e octet : format des données à afficher (1 = alphanumérique, 2 = hexadécimal, 4 = numérique, 8 = numérique avec signe).
- 3e octet : nombre de positions à afficher (00h = tout afficher) ; notons qu'en format numérique, les zéros inutiles seront supprimés automatiquement.
- 4e octet : position du point décimal (00h s'il n'y en a pas).
- 5e octet : durée d'affichage (1 à 254 intervalles de 0,2 s) ou 0 pour un affichage jusqu'au retrait de la carte.
- 6e octet : 1 si d'autres données attendent de pouvoir être affichées, 0 s'il n'y en a plus.
- 7e au 16e octet : données à afficher.

Le cas le plus simple correspond à l'affichage d'un seul bloc de données (6e octet à 00h), qui se fera dès que la carte aura répondu à la commande reçue du lecteur.

Il en sera ainsi, par exemple, si l'on se contente de lire le nombre de points que contient une carte de fidélité, ou encore le solde d'un PME.

Mais dans la plupart des cas, l'application « carte » sera plus bavarde : si elle indique au lecteur (en mettant le 6e octet à 01h) qu'elle a encore quelque chose à afficher, alors le lecteur enverra une nouvelle commande (dont le premier octet sera incrémenté d'une unité) dès que la durée d'affichage requise aura expiré.

La carte répondra alors à cette commande en surchargeant de nouveau sélectivement le bloc de données reçu, et le lecteur mettra son affichage à jour.

Ce processus peut se répéter à l'infini, avec ou sans changement de format d'affichage d'une commande à l'autre.

Bien que notre application PRSPY.BAS tienne en quelques lignes de ZCBasic, il convient de ne pas sous-estimer ses talents (voir dossier « POCKET » de notre cédérom).

Chargée, après compilation, dans une ZC 3.9 (PRSPY39.IMG) ou une ZC 3.3 (PRSPY33.IMG), elle transforme celle-ci en un redoutable outil d'investigation !

Insérée dans n'importe quel terminal acceptant les cartes à puce T = 1 (par exemple Monéo, nous y voila !), la carte enregistre l'en-tête (CLA, INS, P1, P2, Lc ou Le) de toutes les commandes qu'elle en reçoit.

Cela à concurrence de 100 commandes, mais la capacité EEPROM disponible permettrait de faire bien davantage si nécessaire.

Il est, en effet, navrant de constater combien d'applications développées « professionnellement » peuvent enchaîner « en aveugle » jusqu'à des dizaines de commandes, sans se formaliser le moins du monde des réponses, même aberrantes, qu'elles peuvent recevoir !

Il faut dire que notre carte retourne systématiquement un compte rendu de bonne exécution (SW1SW2 = 9000h par défaut), et un bloc de données… vide.

Les choses seraient (tout de même !) différentes si l'on faisait en sorte que la carte réponde « classe d'instruction inconnue » (SW1SW2 = 6E00h).

De même, chacun est libre de choisir à sa guise (et selon l'investigation à effectuer) le contenu des « caractères historiques » de la réponse au reset (première ligne du code source).

Une fois la carte retirée du terminal, on se servira du « Pocket Reader » pour prendre connaissance, sans équipement lourd, de ce qu'elle aura ainsi enregistré.

Il s'affichera `-RST------` à chaque fois que le terminal aura remis la carte à zéro (lors d'une ou plusieurs sessions successives), `0000000000` lorsque rien n'aura été enregistré, ou bien les cinq octets (soit dix caractères hexa) d'en-tête des commandes reçues. Attention, toutefois, à ne pas confondre « b » avec « 6 », ou bien « B » avec « 8 » !

Chaque bloc de données lu étant aussitôt effacé, la carte est immédiatement prête à servir de nouveau dès que son contenu a été visualisé, autant que possible en entier.

Comme nous le laissons entendre depuis un moment, il est intéressant d'expérimenter cette carte sur des lecteurs destinés au porte-monnaie électronique « Monéo ».

En effet, celui-ci fonctionne en T = 1, conséquence directe de l'emprunt, par notre pays pourtant « leader de la carte à puce », de la technologie vieillissante de la « GeldKarte »… allemande.

Cela se voit fort bien dans la réponse au reset, avec l'utilitaire « Analyser.exe » (dossier « ACS » du cédérom), qui pourrait afficher, par exemple, un rapport ressemblant à ceci :

```
ATR Analyser : Developed by Advanced Card Systems Ltd.
ATR : 3B E2 00 FF 81 31 20 75 90 00
TS : 3B = Direct convention
T0 : E2 = ,TB1,TC1,TD1,2 bytes historical bytes follow
TB1 : 00 = II=0, P1=0
TC1 : FF = N=255
TD1 : 81 = ,,,TD,T=1 protocol is supported
TD2 : 31 = TA,TB,,,T=1 protocol is supported
TA3 : 20 = IFSC=32
TB3 : 75 = CWT=18 work etu, BWI=7
----  F=372* D=1* I=25 P=RFU N=minimum ---- ('*'=default)
T01 : 90 : The meaning of these historical bytes are proprietary
T02 : 00 :
---- End of Report ----
```

Le lecteur de poche XL 2500, pour sa part, semble considérer « par défaut » que toute carte T = 1 qui lui est présentée est un PME « Monéo ».

Comme on peut se tromper !

Mis en présence de notre BasicCard « PRSpy », il lui transmettra « les yeux fermés » la suite d'en-têtes de commandes suivante :

```
00 A4 04 0C 06
00 B2 01 64 04
00 B2 01 BC 16
```

Avant même d'avoir eu la moindre carte « Monéo » entre les mains, on sait donc déjà par quel bout commencer son exploration : elle reconnaît la classe 00h (la plus courante), et les instructions normalisées A4h (Select) et B2h (Read Record).

On se doute qu'il ne sera pas bien difficile de pousser plus loin les investigations, par exemple avec notre « espion de cartes asynchrones ».

Ce sera un peu plus compliqué qu'en T = 0, mais tous les détails du protocole T = 1 figurent dans le manuel du kit BasicCard, dont notre cédérom contient une version PDF.

En attendant, voyons un peu ce qu'enregistre notre carte dans un lecteur de GeldKarte (pour PC) de technologie 100 % allemande : pas moins de quinze commandes, cette fois, à commencer par :

```
00 A4 00 0C 00
00 A4 04 0C 09
00 A4 00 0C 00
00 A4 02 0C 02
00 B2 01 04 08
```

Il y a, à tout le moins, une certaine similitude !

Mais le plus étonnant, c'est qu'à aucun moment le logiciel ne se rend compte qu'il n'est pas en présence d'une carte authentique : il affiche des écrans entiers qui en disent long sur tout ce que doit contenir (et sans doute retransmettre à qui de droit) ce genre de PME.

De quoi comprendre un peu mieux l'extrême réticence qui accompagne la mise en place d'un tel « gadget bancaire », qui semble aller très exactement à contre-courant des attentes de ses utilisateurs potentiels.

7 LE CÉDÉROM DU LIVRE

Le cédérom accompagnant cet ouvrage rassemble tous les logiciels développés par l'auteur, les tracés des circuits imprimés des montages proposés, ainsi que de puissants outils de développement offerts par les partenaires qui nous ont aidés à concrétiser ce projet.

Répertoire ACS

On trouvera ici toute la documentation relative aux lecteurs ACR 20 et ACR 30 d'ACS (http://www.acs.com.hk), ainsi que plusieurs versions de leurs drivers, aussi bien PC/SC que « propriétaires ».

Accompagné de son code source en langage C, l'utilitaire ANALYSER.EXE permet d'analyser la « réponse au reset » (ATR) de toute carte asynchrone insérée dans un ACR 20 série, ou dans un « CyberMouse ».

Précisons que le lecteur doit fonctionner en mode « propriétaire » : si son driver PC/SC a été installé, on ne branchera le lecteur sur le port série qu'après le démarrage réussi de Windows.

Répertoire BasicCard

Ce dossier contient l'intégralité du kit de développement BasicCard (dont le langage ZCBasic), et sa documentation en format PDF.

L'ouverture du fichier BasicCardKit.msi déclenche l'installation de la version 4.52 du kit logiciel de la BasicCard, pourvu que l'utilitaire « Microsoft Installer » soit déjà installé sur le PC.

À défaut, InstMsiA.exe et InstMsiW.exe (selon votre système) sont fournis pour procéder au préalable à cette installation.

Le fichier bck4_32.zip contient, pour sa part, la version 4.32, celle-là même dont l'auteur s'est servi pour développer les applications PC/SC offertes sur ce cédérom.

Les versions suivantes seront disponibles, au fur et à mesure de leur sortie, sur http://www.basiccard.com, en téléchargement gratuit.

Répertoire CardEasy

Ce dossier contient (dans le sous-dossier « Setup ») la version complète du logiciel CardEasy offert par ACS, spécifiquement destiné aux lecteurs de cartes à puce « CyberMouse », ACR 20, ou ACR 30 de cette marque (on ne tentera donc pas de l'utiliser, par exemple, avec un lecteur « maison » !).

Une documentation présentant ces excellents lecteurs est fournie dans le sous-dossier « Doc ».

Répertoire CB

Il y a là le résultat de plus de dix ans d'expérimentation autour des cartes bancaires :

- un jeu de programmes d'exploration, écrits en ZCBasic, pouvant être utilisés séparément ou appelés à tour de rôle par CB.BAT ;
- une application Windows de lecture de l'historique des transactions sur un lecteur PC/SC (sous-répertoire « Historique »). Notons que son code source complet est fourni, dans le sous-dossier « Delphi » ;
- différents programmes ZCBasic permettant de procéder aux expériences d'authentification du chapitre 6 (sous-répertoire « Auth »).

Répertoire CyberMouse

Ce répertoire contient la documentation et les drivers du lecteur « CyberMouse » équipant les kits BasicCard, ou disponible séparément (http://www.hitechtools.com).

Répertoire Espasync

Ce dossier rassemble les plus récentes versions du logiciel de l'espion de cartes asynchrones du chapitre 2, aussi bien sous la forme de code source Basic (fichiers .bas) que d'exécutables MS-DOS (fichiers .EXE).

Quatre variantes sont fournies, destinées respectivement aux ports série COM1: ou COM2: du PC, et aux cartes à convention directe ou inverse.

Répertoire PCB

Les tracés de tous les circuits imprimés des montages proposés sont fournis dans le format « .PCB » propre au logiciel BOARD-MAKER ayant servi à les dessiner.

Le sous-répertoire « BMAKER1 » contient une version limitée de cette application MS-DOS, permettant de visualiser et imprimer, avec ou sans modifications, tout ou partie de ces fichiers.

Répertoire PCSC

Ce dossier réunit différentes applications compatibles avec n'importe quel lecteur de cartes à puce certifié PC/SC (CyberMouse, ACR 20 ou 30, etc.).

SCard.exe, en particulier, permet de dialoguer par commandes ISO 7816 avec n'importe quelle carte asynchrone, à protocole T = 0 ou T = 1, quelle que soit la marque du lecteur.

Dans le sous-répertoire « Delphi », on trouvera le code source de l'application « PCSCatr.exe » aimablement offerte par ACS, qui constitue une excellente introduction au développement d'authentiques applications PC/SC sous Delphi.

Dans le sous-dossier « CLASSINS », se trouvent les différents éléments (code source ZCBasic et exécutables compilés) du « scanner » de commandes présenté au chapitre 2.

Dans le répertoire « AUTH », enfin, on trouvera les applications « carte » et « terminal » utilisées dans les expériences d'authentification « interne » et « externe » suggérées au chapitre 2.

Répertoire Pocket

C'est ici que l'on trouvera le nécessaire pour transformer une ZC 3.3 ou une ZC 3.9 en « espion » de terminal T = 1, susceptible d'être relu avec le lecteur de poche « Pocket Reader » que contiennent les kits BasicCard du commerce (http://www.hitechtools.com).

Répertoire SIM

Outre la toute dernière version (Phase 2+) de notre émulateur « BasicSIM », destinée à être programmée dans une BasicCard ZC 5.4 (Rev. A uniquement) ou ZC 5.5 (Rev. B), ce dossier contient notre application ZCBasic d'exploration de cartes « SIM Toolkit » (code source STKM.BAS et exécutable STKM.EXE).

Un sous-dossier « Auth » contient le programme (code source et exécutable) destiné aux expériences d'authentification proposées au chapitre 4.

Répertoire SMS

Dans ce répertoire figure tout le nécessaire pour charger, dans une carte SIM en cours de validité, toutes sortes de mini-messages « spéciaux » (y compris des mélodies de sonnerie !), qui pourront

être envoyés au moyen d'un simple téléphone portable compatible.

Le sous-répertoire « Godsave » contient quelques exemples d'une mélodie bien connue, traduite en différents formats par un système automatisé qui était encore en développement lors de la rédaction de ce livre : on lui pardonnera donc d'éventuelles « fausses notes » !

Dans le sous-dossier « MQP », on trouvera le contenu de la disquette de démonstration d'un programmateur d'EPROM du commerce, dont l'installation met à notre disposition un excellent éditeur hexadécimal (PDED.EXE).

À condition de respecter les conditions d'utilisation de ce shareware (dont on prendra connaissance dans le fichier READ.ME), on tient là un outil très puissant pour opérer, notamment, sur le contenu des SMS que l'on peut être amené à aller lire dans telle ou telle carte SIM (voir notre ouvrage *Téléphones portables et PC*, dans cette même collection).

Répertoire SyncPCSC

Ce dossier rassemble tous les éléments logiciels du projet d'adaptateur PC/SC pour cartes synchrones du chapitre 3.

Répertoire Termin

Dans ce dossier se trouvent tous les logiciels des lecteurs de cartes à puce autonomes ou « serrures électroniques » dont le chapitre 1 décrit la réalisation.

Les programmes pour microcontrôleurs PIC ont été développés en Basic, à l'aide de versions gratuites (7.00 et 7.10) du compilateur LET BASIC (que l'on pourra installer en ouvrant, respectivement, Setup700.exe ou Setup710.exe).

Le manuel de ce puissant outil (dont les versions complètes sont distribuées par Selectronic) est également offert, sous la forme d'un fichier PDF à ouvrir avec Acrobat Reader.

Répertoire Vitale

Il y a là tout le nécessaire pour faire plus ample connaissance avec les cartes « Scot » en général, et avec la carte Vitale en particulier, selon la démarche suggérée au chapitre 5.

Dans le sous-dossier « Auth », on trouvera le programme (code source ZCBasic et exécutable) nécessaire pour procéder aux expériences d'authentification proposées.

Quel ordinateur utiliser ?

Le kit BasicCard et les programmes pour lecteurs PC/SC ne peuvent fonctionner que sous Windows 95 ou supérieur, le maximum de confort étant obtenu en ne dépassant pas Windows 98 ou, à la rigueur, Me (XP déconseillé).

C'est dire qu'un PC équipé d'un processeur 486 DX 100 MHz ou davantage peut suffire, même si un Pentium cadencé à au moins 166 MHz est préférable.

Quelques applications de ce livre fonctionnent exclusivement sous MS-DOS (3.30 ou supérieur).

On en tirera les meilleures performances en quittant complètement Windows, ou même en démarrant carrément l'ordinateur à partir d'une disquette DOS « bootable ».

Il y a là une bonne occasion de faire reprendre du service à de « vieux » PC dépourvus de Windows : notre « espion de cartes asynchrones », en particulier, fonctionne encore honorablement sur un 8088 sans disque dur, cadencé à… 4,77 MHz !

Il conviendra alors de copier les exécutables nécessaires, à partir du cédérom, vers une simple disquette.

Cette possibilité ne devrait pas être dédaignée, car elle pourra rendre service lorsque telle ou telle expérience nécessitera de mettre à contribution plusieurs PC simultanément.